

POLITECNICO DI TORINO

Dipartimento di Elettronica e Telecomunicazioni

Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale



Design of FPGA IP for modular architectures on VirtLAB board

Relatore: prof. Massimo Ruo Roch

Laureando: Andrea Bononi

ACKNOWLEDGMENTS

First and foremost, I would like to thank my parents and my entire family for their unwavering support along the way. I am extremely grateful to my academic advisor and to all the professors that have been part of my academic journey. A special thanks goes also to my girlfriend and to all my friends, who have always believed in me.

CONTENTS

1	INTRODUCTION	4
2	SPECIFICATIONS	6
2.1	Avalon Memory Mapped Interface	6
2.2	HyperRAM Interface Specifications	7
2.2.1	Command-Address	9
2.2.2	Configuration Registers	10
2.2.3	Deep Power Down Mode	11
2.2.4	Power-Up	11
2.3	Converter Design Specifications	12
3	TEST ENVIRONMENT	13
4	DESIGN PARTITIONING	14
4.1	Readdata Converter	17
4.1.1	Execution Unit	21
4.1.2	Voter	21
4.1.3	Control Unit	22
4.2	Synchronizer	22
4.3	CA Builder	25
4.4	Writedata Converter	26
4.5	Configuration Builder	26
4.6	CA Unpacker	27
4.7	Address Generator	27
4.8	Control Unit and Timing Diagrams	28
4.8.1	Memory Timing Constraints	31
5	TEST RESULTS	32
5.1	Preliminary Simulation	32
5.2	Final Simulation	35
5.3	Test on VirtLAB	36
6	FUTURE EXTENSIONS	37
7	VHDL DESCRIPTION	38

ACRONYMS

CR0 Configuration Register 0

CR1 Configuration Register 1

DDR Double Data Rate

DUT Device Under Test

FPGA Field Programmable Gate Array

FSM Finite State Machine

IP Intellectual Property

MCU MicroController Unit

RAM Random Access Memory

SDR Single Data Rate

VCR Virtual Configuration Register

INTRODUCTION

The recent SARS-CoV2 pandemic put a great strain on university courses. Despite the access to physical infrastructures was prohibited, videoconferencing and recorded videos allowed to proceed with the lectures without too many troubles. However, engineering teaching should also involve real laboratory experiences to provide students fundamental skills. When it comes to electronic lessons, it was usually not possible to provide the students the majority of the required instruments (such as digital oscilloscopes, signal generators and spectrum analyzers) given their high cost.

In this scenario, a low-cost experimental printed circuit board, namely the VirtLAB board, was developed at Politecnico di Torino to provide electronic students access to physical devices. Its architecture can be divided in two main sections [1]:

- **User section:** it contains an MCU (STM32L496) and an FPGA (Intel Cyclone 10 LP), which can be easily programmed by the students for educational purposes, together with some LEDs and some switches.
- **Master section:** it contains an MCU (STM32L496), an FPGA (Intel Cyclone 10 LP) and two external memories (a HyperRAM and a QSPI flash) to be used as generic data storage. From the point of view of a student, this side comes already programmed to provide a virtual replacement of the bench equipment.

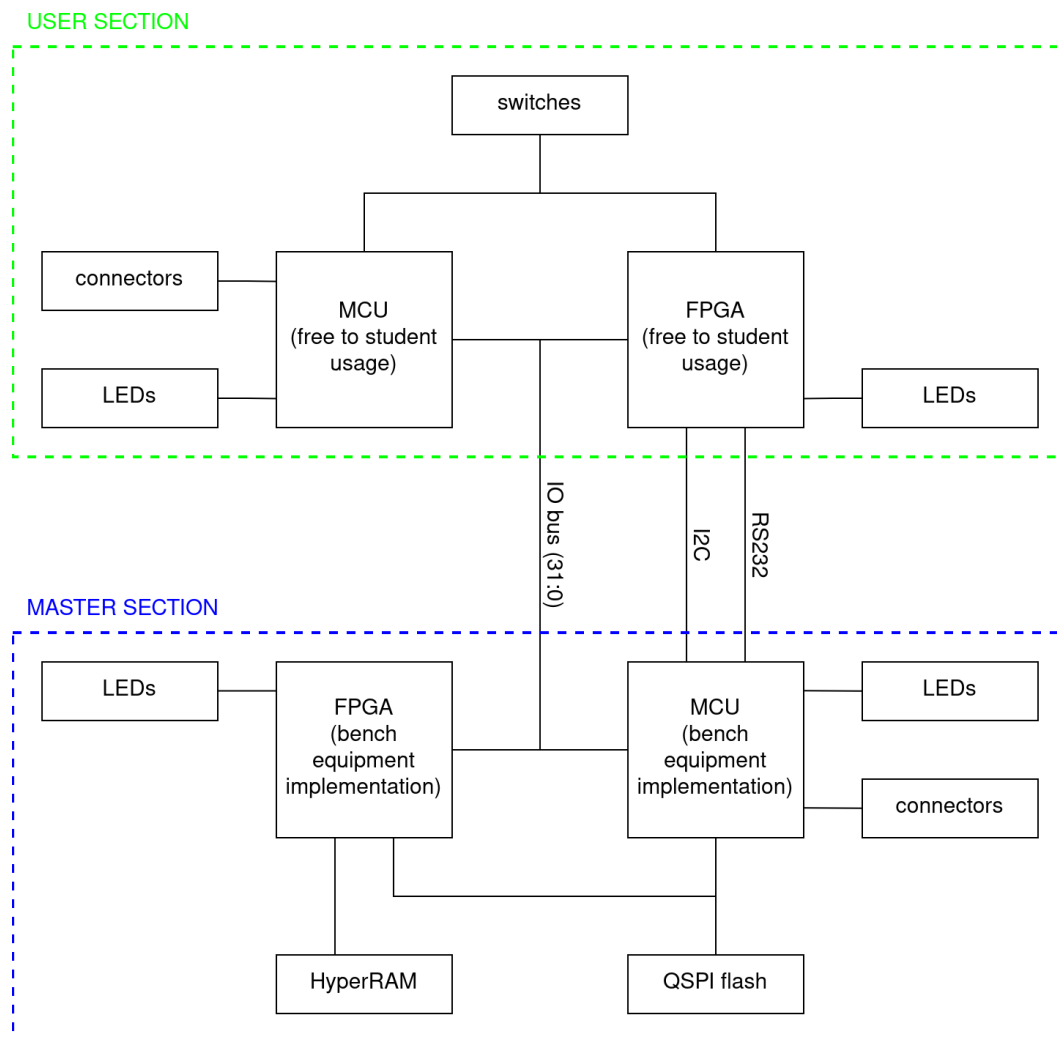


Figure 1.1: VirtLAB board block diagram

Currently, it is not possible to exploit the master section to its full potential. In particular, the FPGA could be used to implement several useful applications. In this regard, the best approach would be to create modular architectures using generic IP cores that share a common communication protocol, namely the Intel Avalon interface. In this way, it would be possible to make full use of the features provided by the Intel CAD software:

- Several general-purpose IP cores with an Intel Avalon interface are already provided by Intel, such as on-chip memories, processors and so on.
- The Intel CAD software is able to automatically create the interconnection logic among IP cores that use an Intel Avalon interface.

At the moment, it is not possible to create an Avalon-based modular architecture able to communicate with any of the external memory storage devices. Indeed, both the HyperRAM interface and the QSPI interface are quite different from the Intel Avalon interface. This paper deals with the design, development and testing of a custom IP core able to convert the HyperRAM interface into an Intel Avalon interface, so that it can be easily managed by any modular architecture. The whole document refers to a HyperRAM model S27KL0641DA, i.e. the exact model employed in the VirtLAB board.

SPECIFICATIONS

2.1 Avalon Memory Mapped Interface

The Avalon interface family defines many interfaces for different applications [3]. What really matters for our purposes is the Avalon Memory-Mapped interface, an address-based read/write interface typical of Host-Agent connections.

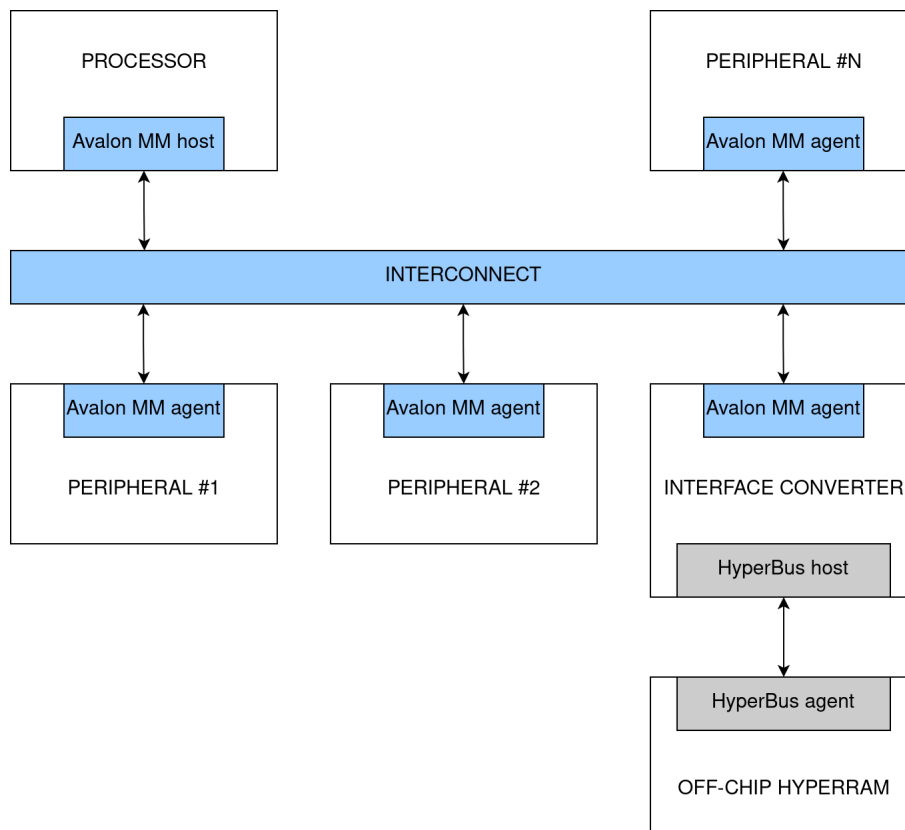


Figure 2.1: Typical Host-Agent system using components with an Avalon Memory Mapped interface (highlighted in light blue). The HyperRAM can be connected to the system only by using a suitable interface converter.

The Avalon Memory Mapped interface includes some always-required signals and several optional signals that might be useful depending on the peripheral. In our case, some specific considerations have to be taken into account:

- In general, the number of clock cycles required to read/write the HyperRAM is variable.
- The system must support burst operations.

Consequently, the Avalon Memory Mapped interface must include the following signals:

- *address*: the address to work with.
- *read*: it is asserted to indicate a read transfer.
- *write*: it is asserted to indicate a write transfer.
- *readdata*: the data read from the agent as a result of a a read transfer.

- *writedata*: the data to be written during a write transfer.
- *readdatavalid*: when asserted, it indicates that the readdata signal contains a valid data.
- *burstcount*: it indicates the number of transfers of a burst operation.
- *waitrequest*: it is asserted by the agent when it is unable to respond to a read/write request.

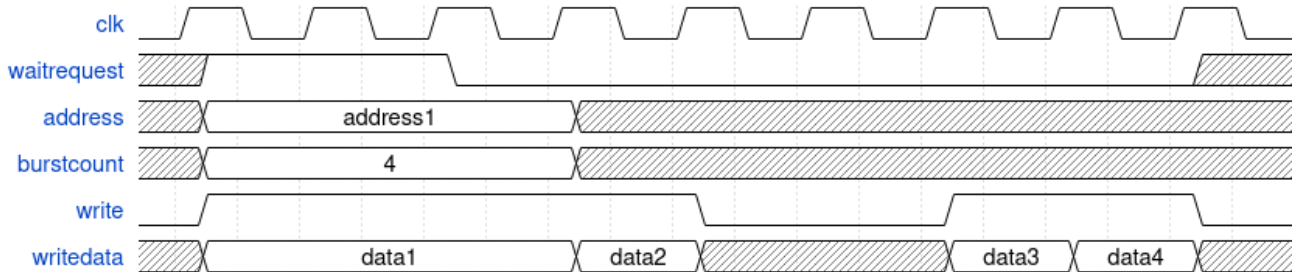


Figure 2.2: Avalon Memory Mapped interface - write operation timing diagram

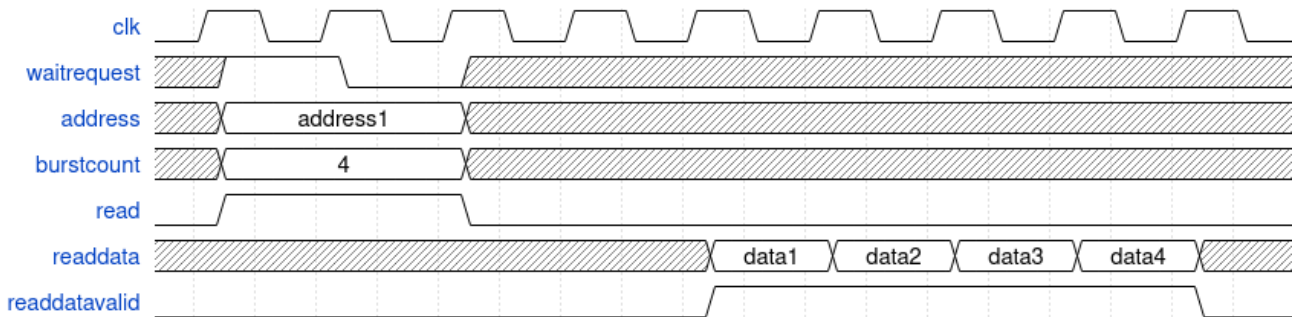


Figure 2.3: Avalon Memory Mapped interface, read operation timing diagram

2.2 HyperRAM Interface Specifications

The HyperRAM interface [2] is based on an 8-bit DDR data bus used to transfer data, addresses and commands. The memory contains a couple of configuration registers that can be written in the same way as the memory locations, but using dedicated addresses. The interface includes the following signals:

- *CK*, *CK#* : differential clock.
- *RESET#* : active-low hardware reset.
- *CS#* : active-low chip select.
- *DQ*: 8-bit IO bus for data, addresses and commands.
- *RWDS*: read/write data strobe with the following functionality:
 - During a read data transfer it is edge-aligned with DQ and it can be used to sample it.
 - During a write data transfer it works as data masking signal.
 - During a command transfer it indicates if additional latency is required.

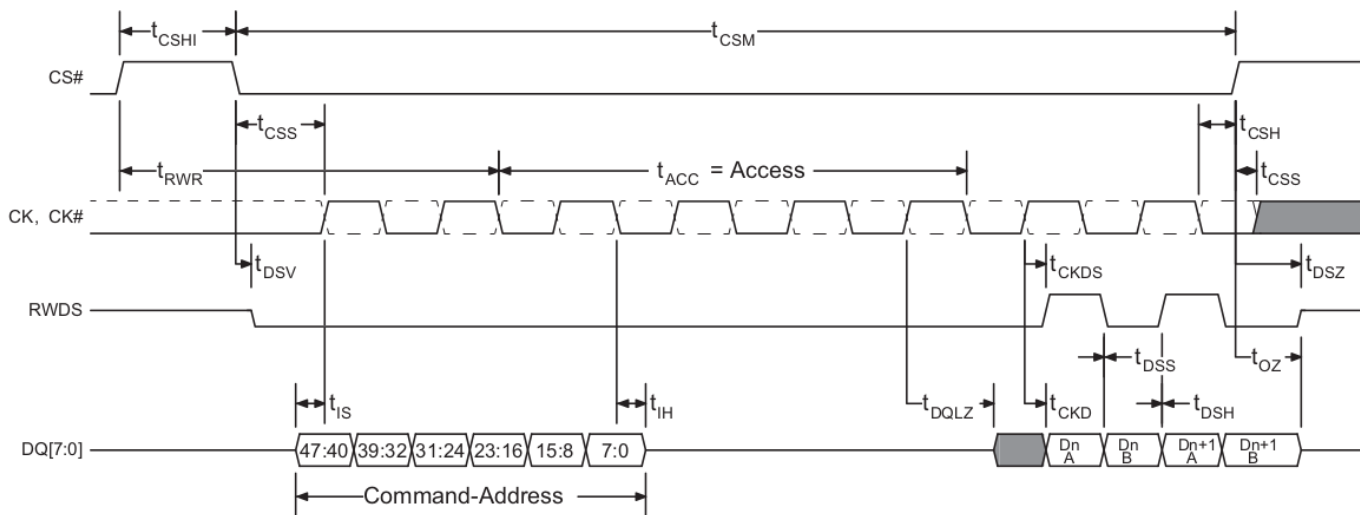


Figure 2.4: HyperRAM interface, read operation timing diagram. During the data transfer, the memory drives both DQ and RWDS. During the command transfer, the host drives DQ and the memory drives RWDS: if RWDS is driven low, the access time is equal to t_{acc} as shown, otherwise the access time is doubled.

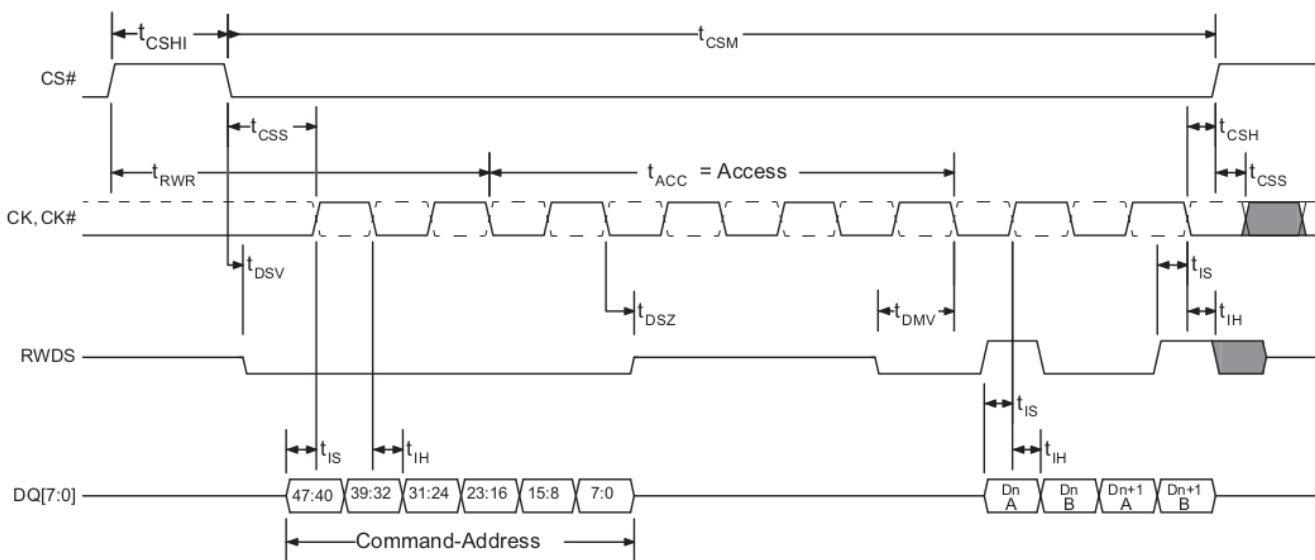


Figure 2.5: HyperRAM interface, write operation timing diagram. During the data transfer, the memory drives both DQ and RWDS. During the command transfer, the host drives DQ and the memory drives RWDS: if RWDS is driven low, the access time is equal to t_{acc} as shown, otherwise the access time is doubled.

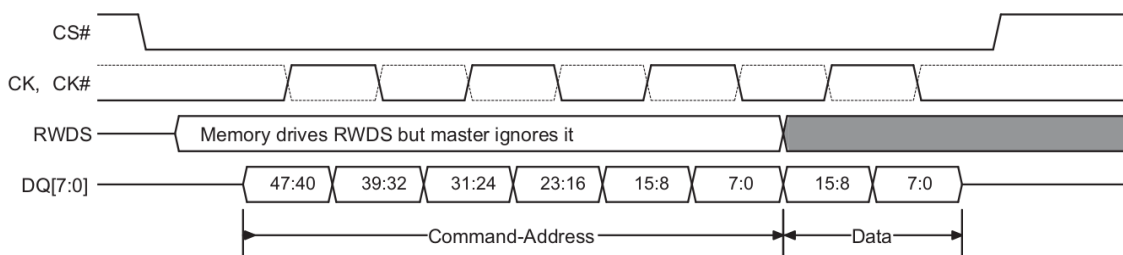


Figure 2.6: HyperRAM interface, register write operation timing diagram. DQ is always driven by the host.

Parameter	Symbol	100 MHz		Unit
		Min	Max	
Chip Select HIGH Between Transactions	t_{CSHI}	10.0	–	ns
HyperRAM Read-Write Recovery Time	t_{RWR}	40	-	ns
Chip Select Setup to next CK Rising Edge	t_{CSS}	3	–	ns
Data Strobe Valid	t_{DSV}	–	12	ns
Input Setup	t_{IS}	1.0	–	ns
Input Hold	t_{IH}	1.0	–	ns
Access Time	t_{ACC}	40	–	ns
Clock to DQs Low Z	t_{DQLZ}	0	–	ns
HyperRAM CK transition to DQ Valid (64 Mb)	t_{CKD}	1	7	ns
HyperRAM CK transition to DQ Valid (128 Mb)			8	
HyperRAM CK transition to DQ Invalid (64 Mb)	t_{CKDI}	0.5	5.2	ns
HyperRAM CK transition to DQ Invalid (128 Mb)			6.2	
CK transition to RWDS valid (64 Mb)	t_{CKDS}	1	7	ns
CK transition to RWDS valid (128 Mb)			8	
RWDS transition to DQ Valid	t_{DSS}	-0.8	+0.8	ns
RWDS transition to DQ Invalid	t_{DSH}	-0.8	+0.8	ns
Chip Select Hold After CK Falling Edge	t_{CSH}	0	–	ns
Chip Select Inactive to RWDS HI-Z	t_{DSZ}	–	7	ns
Chip Select Inactive to DQ HI-Z	t_{OZ}	–	7	ns
HyperRAM Chip Select Maximum LOW Time - Industrial Temperature	t_{CSM}	-	4.0	us
HyperRAM Chip Select Maximum LOW Time - Industrial Plus Temperature			1.0	us

Figure 2.7: HyperRAM interface timing parameters.

2.2.1 Command-Address

As we saw in section 2.2, the operation command and the address are grouped in a 48-bit block, which is sent to the memory by the host one byte for clock level. Every bit of this block has its own meaning:

CA Bit#	Bit Name	Bit Function
47	R/W#	Identifies the transaction as a read or write. R/W# = 1 indicates a Read transaction R/W# = 0 indicates a Write transaction
46	Address Space (AS)	Indicates whether the read or write transaction accesses the memory or register space. AS = 0 indicates memory space AS = 1 indicates the register space The register space is used to access device ID and Configuration registers.
45	Burst Type	Indicates whether the burst will be linear or wrapped. Burst Type = 0 indicates wrapped burst Burst Type = 1 indicates linear burst
44-16	Row & Upper Column Address	Row & Upper Column component of the target address: System word address bits A31-A3 Any upper Row address bits not used by a particular device density should be set to 0 by the host controller master interface. The size of Rows and therefore the address bit boundary between Row and Column address is slave device dependent.
15-3	Reserved	Reserved for future column address expansion. Reserved bits are don't care in current HyperBus devices but should be set to 0 by the host controller master interface for future compatibility.
2-0	Lower Column Address	Lower Column component of the target address: System word address bits A2-0 selecting the starting word within a half-page.

Figure 2.8: Command-Address (CA) bit assignment

2.2.2 Configuration Registers

The S27KL0641DA HyperRAM contains two configuration registers that allow the user to set up different parameters.

Register	CA Bits	47	46	45	44-40	39-32	31-24	23-16	15-8	7-0
Configuration Register 0 Read		C0h or E0h				00h	01h	00h	00h	00h
Configuration Register 0 Write		60h				00h	01h	00h	00h	00h
Configuration Register 1 Read		C0h or E0h				00h	01h	00h	00h	01h
Configuration Register 1 Write		60h				00h	01h	00h	00h	01h

Figure 2.9: Command-Address configuration to access the configuration registers

CR1 Bit	Function	Settings (Binary)
15-2	Reserved	000000h — Reserved (default) Reserved for Future Use. When writing this register, these bits should be cleared to 0 for future compatibility.
1-0	Distributed Refresh Interval	10b — default 4 μ s for Industrial temperature range devices 1 μ s for Industrial Plus temperature range devices 11b — 1.5 times default 00b — 2 times default 01b — 4 times default

Figure 2.10: Configuration Register 1 (CR1) bit assignment

CR0 Bit	Function	Settings (Binary)
15	Deep Power Down Enable (64 Mb)	1 - Normal operation (default) 0 - Writing 0 to CR[15] causes the device to enter Deep Power Down
	Reserved (128 Mb)	Reserved for 128 Mb dual-die stack
14-12	Drive Strength	000 - 34 ohms (default) 001 - 115 ohms 010 - 67 ohms 011 - 46 ohms 100 - 34 ohms 101 - 27 ohms 110 - 22 ohms 111 - 19 ohms
11-8	Reserved	1 - Reserved (default) Reserved for Future Use. When writing this register, these bits should be set to 1 for future compatibility.
7-4	Initial Latency	0000 - 5 Clock Latency - 133 MHz 0001 - 6 Clock Latency - 166 MHz (default) 0010 - Reserved 0011 - Reserved 0100 - Reserved ... 1101 - Reserved 1110 - 3 Clock Latency - 83 MHz 1111 - 4 Clock Latency - 100 MHz
3	Fixed Latency Enable (64 Mb)	0 - Variable Latency - 1 or 2 times Initial Latency depending on RWDS during CA cycles. 1 - Fixed 2 times Initial Latency (default)
	Reserved (128 Mb)	1 - Fixed 2 times Initial Latency (default)
2	Hybrid Burst Enable	0: Wrapped burst sequences to follow hybrid burst sequencing 1: Wrapped burst sequences in legacy wrapped burst manner (default)
1-0	Burst Length	00 - 128 bytes 01 - 64 bytes 10 - 16 bytes 11 - 32 bytes (default)

Figure 2.11: Configuration Register 0 (CR0) bit assignment

2.2.3 Deep Power Down Mode

The HyperRAM can enter a special mode, called Deep Power Down (DPD) mode, in which the current consumption is driven to the lowest possible level. This mode is entered setting the *Deep Power Down Enable* bit in CR0. The next access to the device, driving *CS#* low then high (dummy transaction), will cause the device to exit the DPD mode, as well as a hardware reset. A certain time is required to enter or exit the DPD mode.

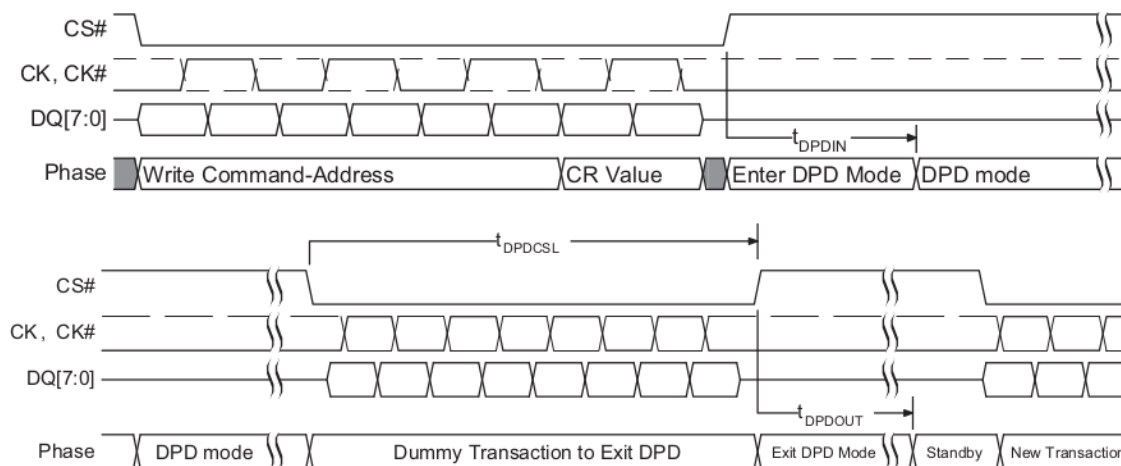


Figure 2.12: DPD timing diagram

2.2.4 Power-Up

The device must not be selected during the power-up, *CS#* has to remain high for a certain time. If *RESET#* is low during the power-up, the time counting does not start until *RESET#* goes high.

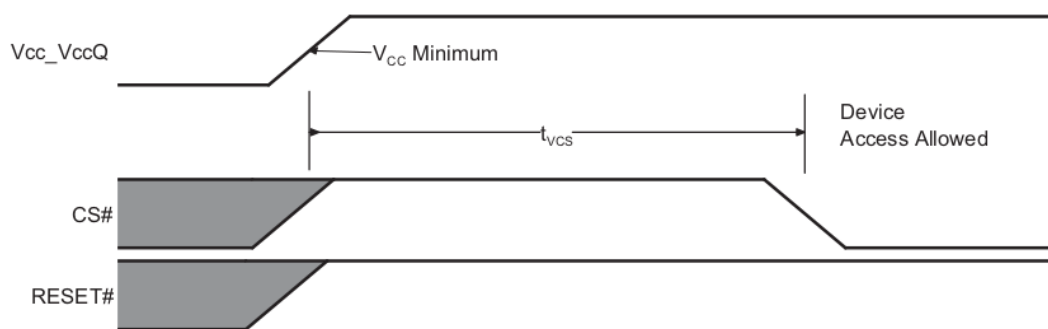


Figure 2.13: Power-up with *RESET#* high

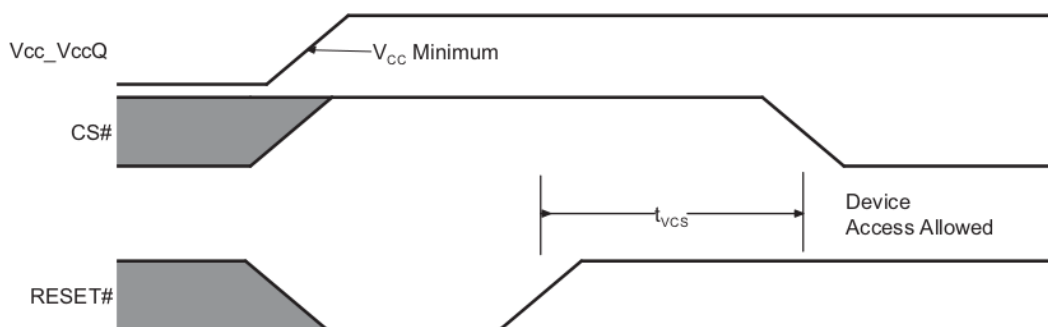


Figure 2.14: Power-up with *RESET#* low

2.3 Converter Design Specifications

Every HyperRAM interface uses a 32-bit addressing. However, the S27KL0641DA device is a 64 Mb memory partitioned in 16-bit words, therefore its addressing takes only 22 bits. On the other hand, the register space access requires dedicated addresses that are not in conflict with the ones related to the memory locations.

In this design, it was decided to virtualize the memory access. To be more precise, the Avalon Memory Mapped interface refers to a 23-bit virtual address, that is translated by the interface converter in the corresponding physical address of the HyperRAM:

- The virtual addresses from 0 to $2^{22} - 1$ correspond to the physical memory location addresses from 0 to $2^{22} - 1$.
- The virtual address 2^{22} refers to a virtual configuration register that allows the user to set up different parameters. From the point of view of the Avalon interface, the host can only access the virtual configuration register, whereas the physical configuration registers of the HyperRAM cannot be accessed. In this way, it is possible to decide which parameters can be dynamically configured and which cannot.
- The virtual addresses from $2^{22} + 1$ to $2^{23} - 1$ are reserved for future expansions.

The 16-bit virtual configuration register (VCR) is organized in the following way:

VCR BIT	FUNCTION	SETTINGS
0	Deep Power Down Enable	0: normal operation, exit DPD mode (default) 1: enter DPD mode
1	Fixed Latency Enable	0: variable latency (default) 1: fixed latency
2-15	Reserved	Reserved for future expansions.

By default, the memory works in normal mode. The host can force it to enter the DPD mode setting the *Deep Power Down Enable* bit in VCR and then to go back to normal mode resetting that same bit. The interface converter shall detect any update of the *Deep Power Down Enable* bit and consequently drive the memory according to figure 2.12. Moreover, since a DPD exit request corresponds to a VCR update, it is possible for the host to update multiple parameters at the same time. For this reason, the interface converter must automatically drive a CR1 update after exiting the DPD mode.

The default configuration of VCR does not match the default configuration of CR1 (figure 2.10). For this reason, the interface converter must automatically update CR1 after the power-up before allowing the host to start a new operation.

As we can see from figures 2.13 and 2.14, the interface converter must wait for the memory to power-up before allowing the host to start a new operation. Every time the memory is reset, it must be powered-up again.

As far as the frequency is concerned, the S27KL0641DA HyperRAM can work up to 100 MHz. However, the interface converter is designed to work at 50 MHz, sending to the memory a clock at that same frequency (as described in section 4.1).

TEST ENVIRONMENT

Before starting the interface converter design, it is necessary to define a test environment for it. We can exploit some of the IP cores provided by the CAD software (which are of course well-functioning) to create an Avalon Memory Mapped system suitable for the test. In particular, the test environment is an processor-based system that read some inputs and change the status of some LEDs according to their value. The HyperRAM is employed as data memory for the processor.

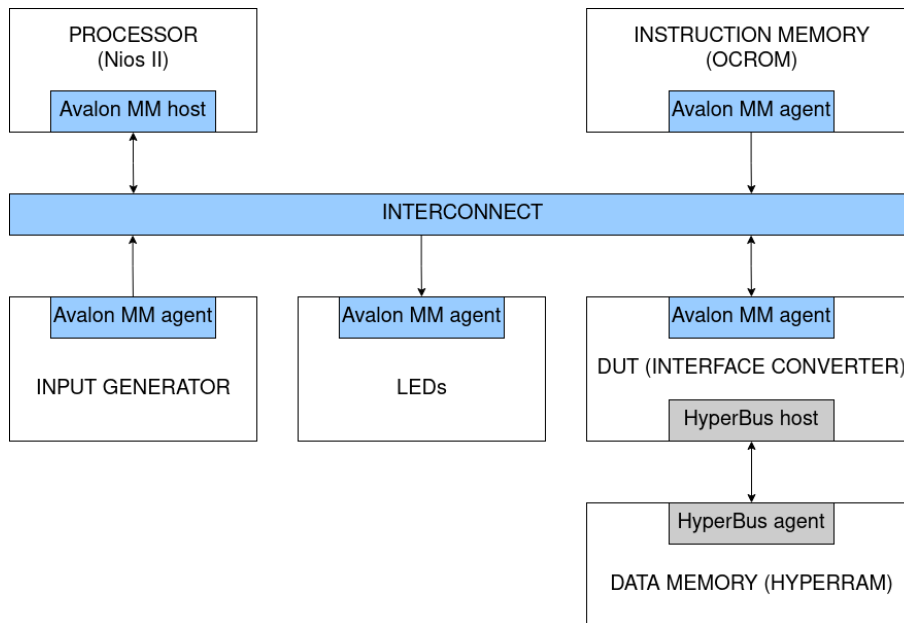


Figure 3.1: Test environment for the interface converter

To ensure that the test environment is well-functioning, the easiest way is to replace the DUT and the HyperRAM with an on-chip RAM, which can be obtained simply by using an IP core provided by the CAD software.

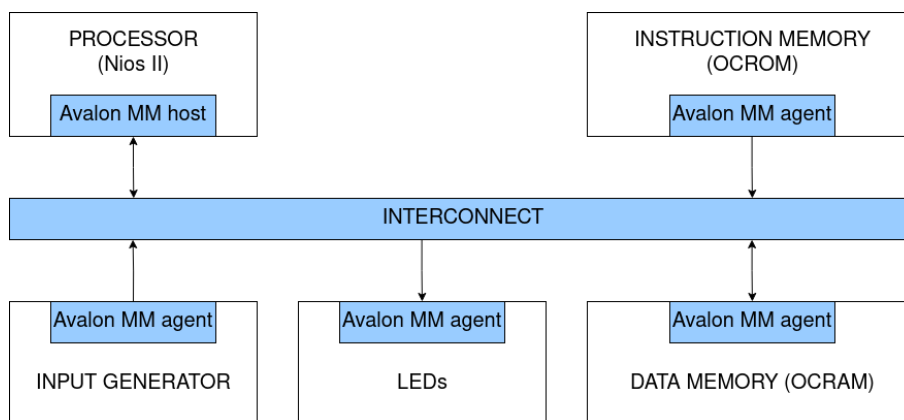


Figure 3.2: Test environment verification

DESIGN PARTITIONING

From now on, the interface converter is referred as *avs_hram_converter*, i.e. the name of the custom IP implementing it. The top-level view of this IP is shown in figure 4.1. On the Avalon side, the address line is on 23 bits and the data line is on 16 bits, as described in chapter 2, section 2.3. The burstcount signal is on 11 bits, i.e the maximum possible parallelism, corresponding to a theoretical maximum burst length equal to 2^{10} as stated in the Avalon documentation.

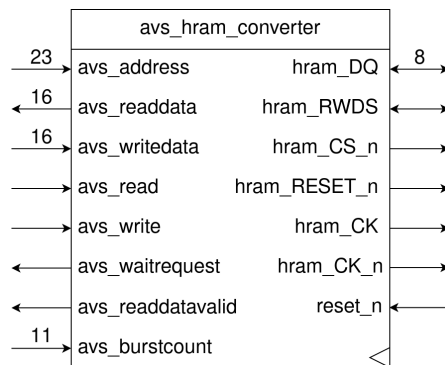


Figure 4.1: Top-level view of the interface converter

Unfortunately, it is usually not possible to push the burst length up to its theoretical maximum, since the duration of any memory operation is upper bounded. Considering that the latency of the converter depends on its implementation, it is not possible to estimate the actual upper bound of the burst length in advance. For this reason, the maximum parallelism is employed and the effective maximum of the burst length will be estimated after completing the design (section 4.8.1).

The design follows a top-down approach. At first, it is important to point out the main features to be implemented:

- **Command-Address building:** the Avalon input signals must be re-organized arranging the CA.
- **Configuration registers building:** every time the virtual configuration register is written, it is necessary to convert its content so that the physical configuration registers of the memory can be properly updated.
- **SDR to DDR conversion:** the 16-bit SDR data provided at the Avalon interface must be converted in an 8-bit DDR data to put it on the memory data bus.
- **DDR to SDR conversion:** the 8-bit DDR data provided by the memory (which is synchronous with RWDS and not with the internal clock) must be converted in a 16-bit SDR data and synchronized with the clock.
- **Clock shifting and clock gating:** the internal clock must be shifted by 90 degrees and properly gated before being sent to the memory.
- **Timer:** the system must be aware the passage of time to satisfy all the timing requirements.
- **Address reconstruction.** As we can see in figure 2.2, the host can interrupt a write operation at any time. However, the HyperRAM does not support this feature. For this reason, the interface converter must end the operation and start a new one when the burst is resumed. The new operation shall begin at the right address, i.e. the one immediately after the last written location.

The CAD software provides an IP implementing a clock controller. Indeed, we can just create a custom IP implementing all the features except the clock gating (*avs_hram_mainconv*) and combine it with the clock controller IP (*clkctrl*) to create the interface converter (*avs_hram_converter*), as shown in figure 4.2. The *avs_hram_mainconv* IP is composed by an FSM-based control unit (section 4.8) and an execution unit (figure 4.3):

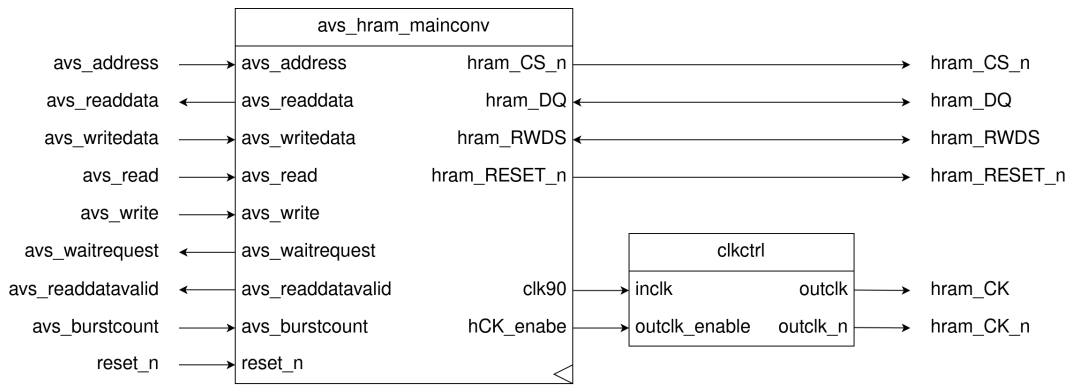


Figure 4.2: Architecture of the *avs_hram_converter* IP

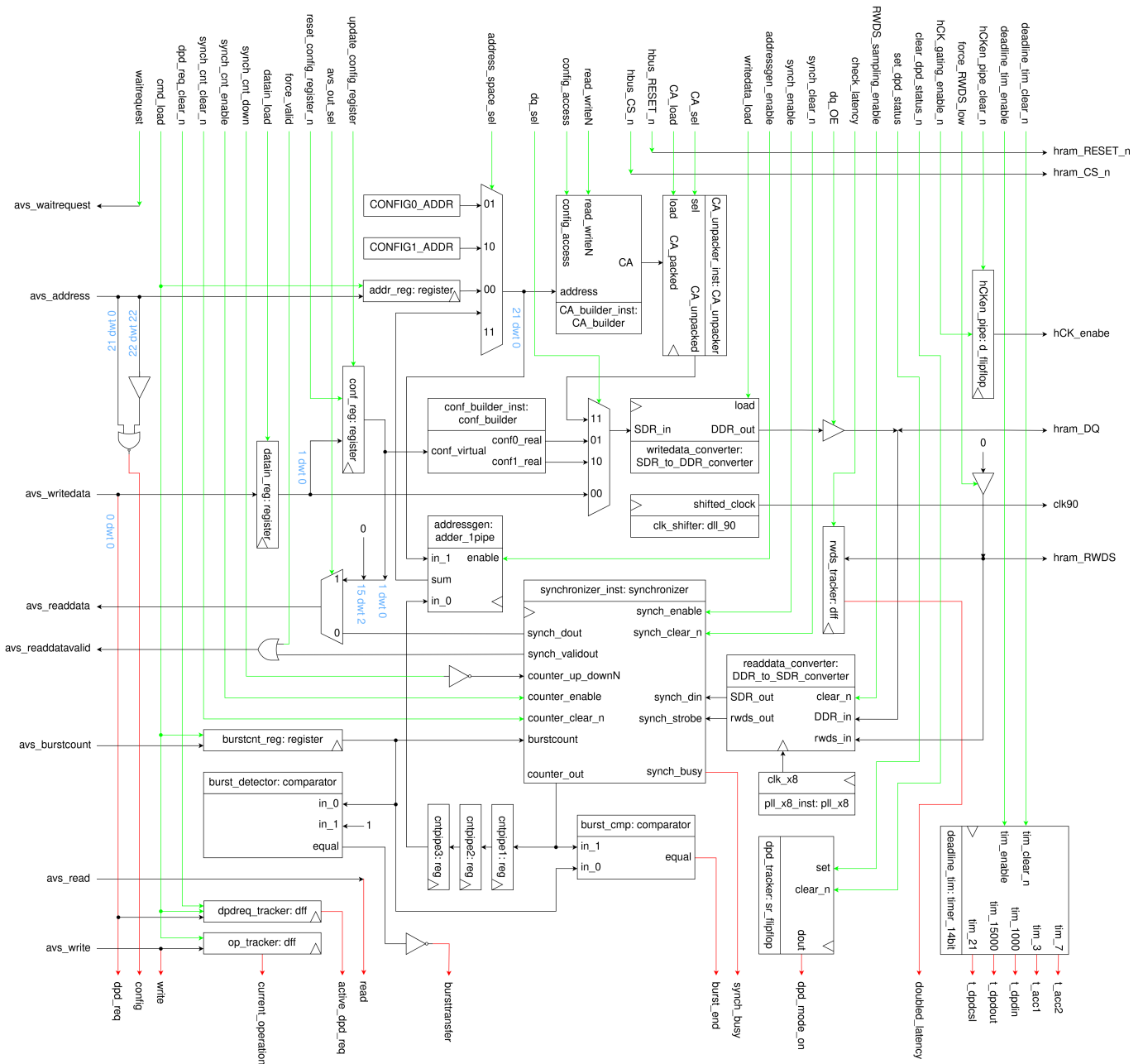


Figure 4.3: Execution unit of the *avs_hram_mainconv* IP. Avalon signals are placed on the left. HyperRAM signals are placed on the right. The top green signals represent the control signals received from the control unit. The bottom red signals represent the status signals sent to the control unit.

The role of each block represented in figure 4.3 is described in the table below:

NAME	TYPE	NICKNAME	DESCRIPTION
<i>addr_reg</i>	<i>register</i>	address register	It stores the virtual address provided on the Avalon side.
<i>datain_reg</i>	<i>register</i>	data register	It stores the input data provided on the Avalon side.
<i>conf_reg</i>	<i>register</i>	configuration register	It implements the virtual configuration register as described in section 2.3.
<i>burstcnt_reg</i>	<i>register</i>	burst length register	It stores the burst length value provided on the Avalon side.
<i>CA_builder_inst</i>	<i>CA_builder</i>	CA builder	It builds the 48-bit Command-Address starting from the address value and the operation type. Refer to section 4.3 for a detailed description.
<i>CA_unpacker_inst</i>	<i>CA_unpacker</i>	CA unpacker	It separates the Command-Address in 3 different 16-bit data. Each of them can be picked out depending on the value of a selector. Refer to section 4.6 for a detailed description.
<i>conf_builder_inst</i>	<i>conf_builder</i>	configuration builder	It generates the content of the physical configuration registers of the HyperRAM starting from the content of the virtual configuration register. Refer to section 4.5 for a detailed description.
<i>writedata_converter</i>	<i>SDR_to_DDR_converter</i>	writedata converter	It converts a 16-bit SDR data in a 8-bit DDR data, so that it can be channeled into the HyperRAM <i>DQ</i> bus. Refer to section 4.4 for a detailed description.
<i>RWDS_tracker</i>	<i>dff</i>	RWDS tracker	It samples the value of RWDS using the internal clock. It is not suitable for sampling it while receiving a DDR data, it is intended to be used during the CA transmission.
<i>dpdreq_tracker</i>	<i>dff</i>	DPD request tracker	It is employed to keep track of a DPD mode entry request.
<i>op_tracker</i>	<i>dff</i>	operation tracker	It is employed to keep track of the type of operation currently in execution.
<i>dpd_tracker</i>	<i>sr_flipflop</i>	DPD tracker	It is employed to keep track of the current memory mode (NORMAL, DPD).
<i>readdata_converter</i>	<i>DDR_to_SDR_converter</i>	readdata converter	It samples <i>RWDS</i> and <i>DQ</i> using a 400 MHz clock, converting the 8-bit DDR data in a 16-bit SDR data. It also provides a version of <i>RWDS</i> with a rising-edge approximately center-aligned with the SDR data. Refer to section 4.1 for a detailed description.
<i>pll_x8_inst</i>	<i>pll_x8</i>	pll x8	It generates a 400 MHz clock starting from the internal 50 MHz clock.
CONTINUED...			

NAME	TYPE	NICKNAME	DESCRIPTION
<i>synchronizer_inst</i>	<i>synchronizer</i>	synchronizer	It samples the 16-bit SRD at the output of readdata converter (using the center-aligned version of RWDS it provides) to synchronize it with the internal clock. It contains an 11-bit up-counter that can be accessed from outside.
<i>burst_detector</i>	<i>comparator</i>	burst detector	It notifies the system when the burst length is greater than 1.
<i>burst_cmp</i>	<i>comparator</i>	burst comparator	It is employed to notice when the burst transmission has reached its end (i.e. the burst length).
<i>addressgen</i>	<i>adder_1pipe</i>	address generator	It is employed when a write burst operation is resumed (after being stopped) to generate the new starting address.
<i>clk_shifter</i>	<i>ddl_90</i>	clock shifter	It introduces a 90 degree delay on the internal clock.

4.1 Readdata Converter

As we can see in figure 2.4, during the data transfer of a read operation the memory drives both *DQ* and *RWDS*, the former being edge-aligned to the latter. The effect of board traces can be neglected considering that the *DQ* trace and the *RWDS* trace have a similar length on the PCB. The main goal of *readdata converter* is to introduce a proper delay on *RWDS* so that it can be used to sample *DQ*; in particular, referring to figure 4.4, *DQ_out* and *RWDS_out* shall be approximately center-aligned.

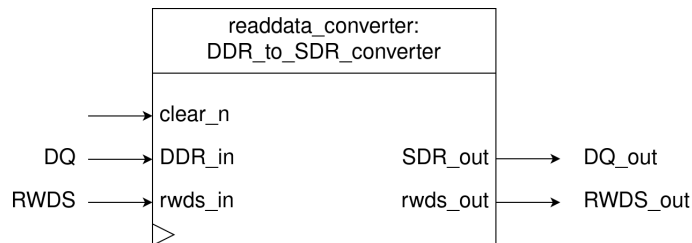


Figure 4.4: Top-level view of *readdata converter*.

Considering that *RWDS* oscillates at the clock frequency, a first possibility would be to introduce a combinational delay. In this way, it would be possible to push the memory frequency up to 100 MHz (assuming that all the other blocks in the system are fast enough). Theoretically, the Intel Cyclone 10 LP FPGA allows to introduce a controlled combinational delay on its input pins; however, the documentation is really poor, especially when it comes to explaining how to do it in the CAD software. Due to the many troubles encountered, it was decided to follow a different approach.

Instead of forcing a combinational delay, a very high frequency clock (oversampling clock) can be employed. In this way, *RWDS* can be shifted using just a flip-flop chain. To do that, the system need first to oversample both *RWDS* and *DQ* to synchronize them with the high-frequency clock. During the oversampling, some samples are collected violating the setup-hold constraints and they must be considered invalid. Indeed, the oversampling frequency must be high enough to collect more valid samples than invalid samples within a semi-period of *DQ* and *RWDS*, even in the worst case scenario.

To compute the minimum sampling frequency, a setup-hold time lower than 3/4 of the clock period has been considered (assumption that shall be verified during the synthesis). Considering a rising-edge sampling, in the worst case

both the first and the last samples are not valid, therefore the oversampling frequency must be at least 10 times the memory frequency:

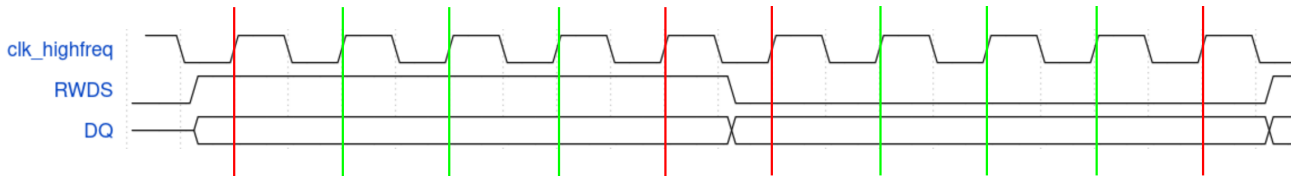


Figure 4.5: Oversampling frequency 10 times the memory frequency, single-edge sampling. The green samples are valid, the red samples are invalid.

Switching to a double-edge sampling, in the worst case three samples are not valid, therefore the oversampling frequency must be at least 8 times the memory frequency:

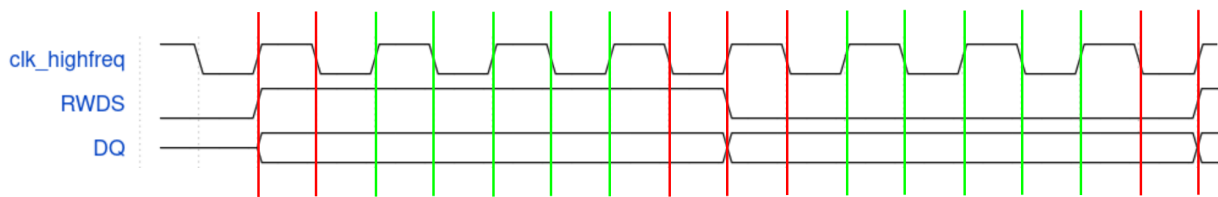


Figure 4.6: Oversampling frequency 8 times the memory frequency, double-edge sampling. The green samples are valid, the red samples are invalid.

The oversampling frequency is generated using a PLL available in the FPGA. As written in the Intel Cyclone 10 LP datasheet, the maximum output frequency of the PLL is equal to 472.5 MHz, therefore the memory frequency is upper bounded at 59 MHz (considering a double-edge sampling). In other words, to make the oversampling technique work it is not possible to push the memory frequency up to its maximum (100 MHz). Indeed, it was decided to lower the memory frequency to 50 MHz and to use a 400 MHz sampling frequency (generated by means of a PLL with a multiplication factor equal to 8).

Another important parameter is the value of the shift to be introduced on *RWDS*. Theoretically, it should be delayed by 1/4 of the clock period (5 ns, i.e. two sampling clock periods) to make it center-aligned with *DQ*. However, in the real case the setup-hold violations must be taken into account. As we can see in figure 4.7, a delay of 5 ns may lead to an oscillation of *RWDS_out* too close to the next variation of *DQ_out*. Indeed, the goal of *readdata converter* is to generate *RWDS_out* such that it can be used to sample *DQ_out* without introducing any setup-hold violation.

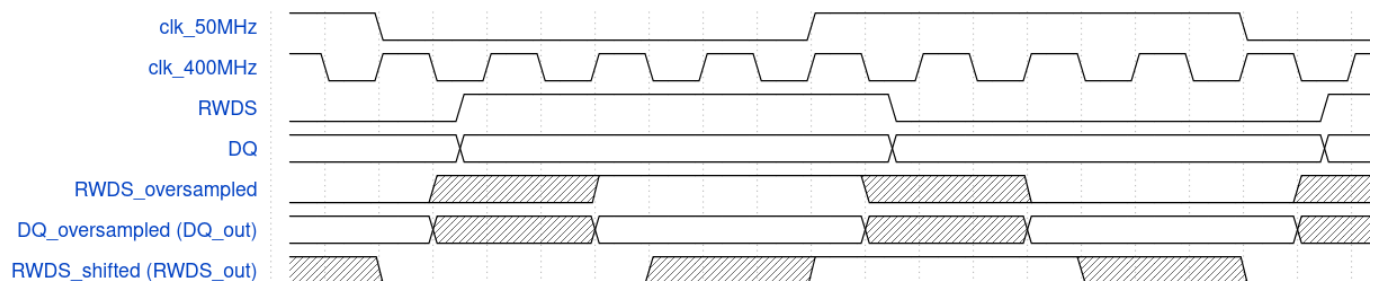


Figure 4.7: Example of possible setup-hold violation during the oversampling. In this case, *RWDS_out* is shifted by two sampling clock cycles with respect to *DQ_out*.

To make sure that no setup-hold violations are present, it is necessary to reduce the shift to a single sampling clock cycle (2.5 ns) as shown in figure 4.8.

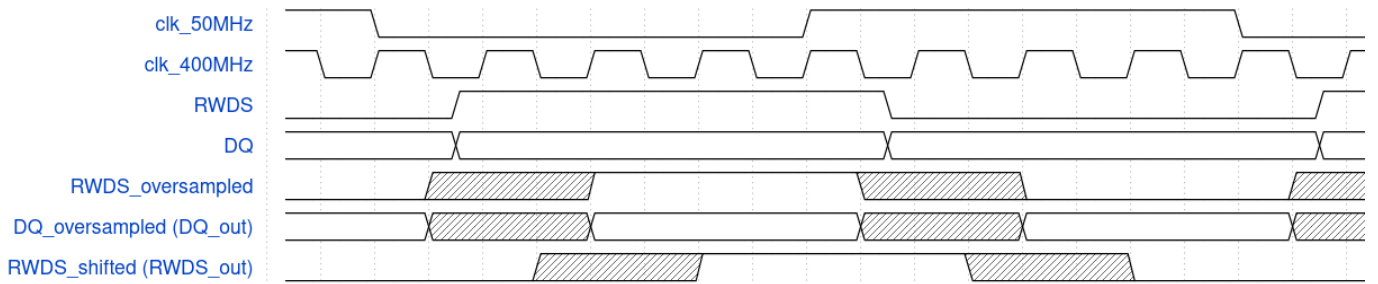


Figure 4.8: Timing diagram with a single clock cycle shift between *RWDS_out* and *DQ_out*.

As we already said, the goal of *readdata converter* is to introduce a shift between *RWDS_out* and *DQ_out* so that the former can be employed to sample the latter. The sampling is implemented outside *readdata converter* by means of a register having the clock pin connected to *RWDS_out* and the input data pin connected to *DQ_out* (referring to figure 4.3, this register is located inside the *synchronizer*). In this regard, it is important to highlight that the timing behavior described in figure 4.8 works correctly only assuming that there are no other delay contributions between *readdata converter* and the sampling register (i.e. that the delay between the clock pin and the input pin of the sampling register is exactly equal to the delay between *DQ_out* and *RWDS_out*). Unfortunately, this assumption cannot be considered true. In general, we must consider the circuit in figure 4.9.

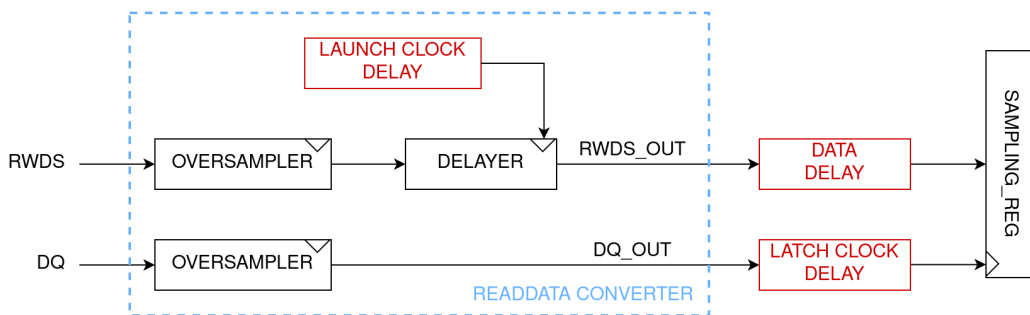


Figure 4.9: Basic scheme of *readdata converter*. All the delay terms are considered.

To make *readdata converter* work, it is necessary to estimate the value of the delay terms and compensate them. A basic timing analysis in which *readdata converter* behaves as described in figure 4.8 is reported in figure 4.10.

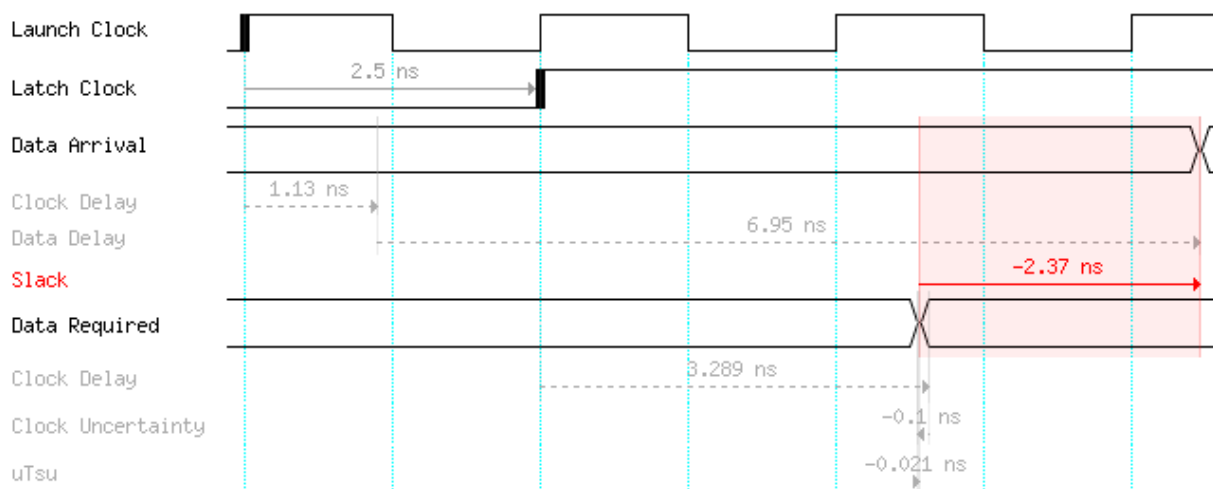


Figure 4.10: Timing analysis according to figure 4.8.

As we can see in figure 4.10, $RWDS_out$ (the latch clock) is delayed by 2.5 ns with respect to DQ_out as expected. However, the value of the input data pin of the sampling register (data arrival) changes 2.37 ns later than required (i.e. slightly less than a period of the high-frequency clock). To compensate this timing violation, *readdata converter* can be designed to introduce a further shift of 2.5 ns (thus, a total delay of 5 ns), as shown in figure 4.11.

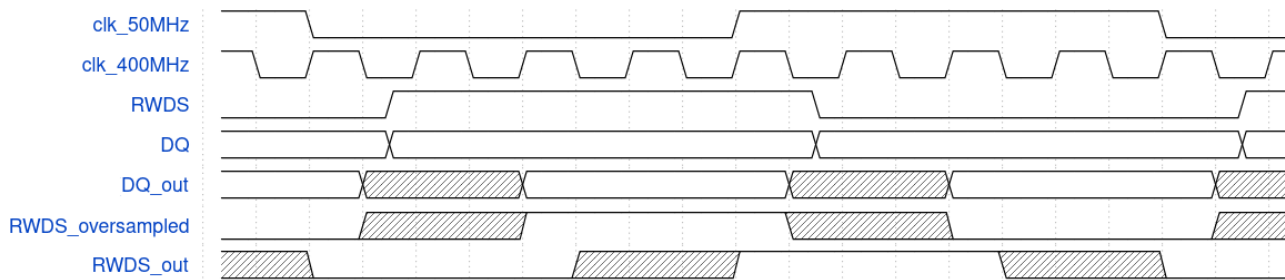


Figure 4.11: *readdata converter* timing diagram able to prevent setup/hold violations.

The timing diagram represented in figure 4.11 is able to prevent setup/hold violation. However, it is not able to make the system in figure 4.9 work. Indeed, $RWDS_out$ cannot just shift $RWDS_oversampled$, since this signal may have spurious oscillations in correspondence of the invalid samples (which may assume whatever logic value). For this reason, the system oversamples $RWDS_in$ and then implements a majority decision: when at least 5 samples out of 8 are different from the current decision about the value of $RWDS_in$, the current decision is toggled. Figure 4.12 represents some different scenarios, showing that the current decision does not display spurious oscillations regardless of the value assumed by the invalid samples. DDR_in is considered valid the clock period corresponding to the decision change; in this way, no setup/hold violations occur. $RWDS_out$ is generated toggling a signal with 2 clock periods delay with respect to the variation of DDR_out , therefore setup/hold violations are prevented and no spurious oscillations are present. Moreover, its value is inverted with respect to the current decision, so that its rising edges correspond to the valid valued of SDR_out .

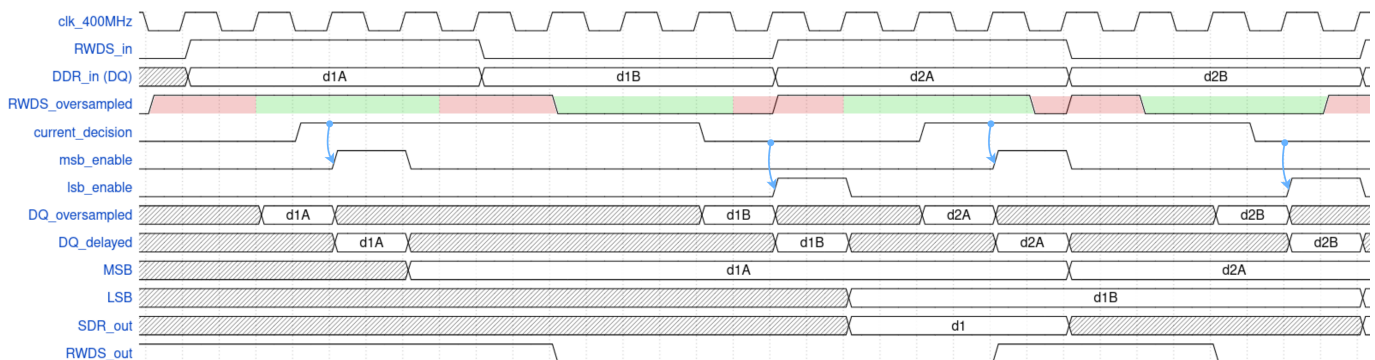


Figure 4.12: Timing diagram of different scenarios during the oversampling. Three samples per $RWDS_in$ period are not valid (worst-case). The values of $RWDS_oversampled$ highlighted in red are invalid (they corresponds to a setup/hold violation, therefore they may assume whatever logic value).

The values of $RWDS_oversampled$ highlighted in green are valid. The current decision is obtained through a combinational circuit and it is sampled by the rising edge of the high-frequency clock.

The architecture of *readdata converter* implementing the timing diagram in figure 4.12 is divided in an execution unit (described in subsection 4.1.1) and a control unit (described in subsection 4.1.3) implemented as a finite-state machine. As we can see in figure 4.13, the execution unit is strongly pipelined to be able to work with a 400 MHz clock frequency. Indeed, the circuit generating the current decision from the collected samples is not combinational, differently from what is assumed in figure 4.12, and the control signals generated by the control unit are pipelined too. Consequently, the actual timing diagram is slightly different from the one shown in figure 4.12, since a certain latency is introduced. However, the working principle remains the same.

4.1.1 Execution Unit

Figure 4.13 represents the architecture of the execution unit of *readdata converter*. The current decision about the value of *RWDS_in* is stored in a dedicated type-T flip-flop, namely the *tracker*. Two different flip-flop chains are used to oversample *RWDS_in* on both rising and falling edges of the high-frequency clock. The majority decision is implemented using a *voter* and some logic gates. In particular, the *voter* is able to notice if the sum of all the collected samples is equal (*eq4*) or greater (*gt4*) than four. If the sum is greater than four (which means that there are more samples equal to logic 1 than to logic 0) and the current decision (stored in the *tracker*) is logic 0, the decision is toggled. If the sum is lower than four (neither *eq4* nor *gt4* are set, thus there are more samples equal to logic 0 than to logic 1) and the current decision is logic 1, the decision is toggled.

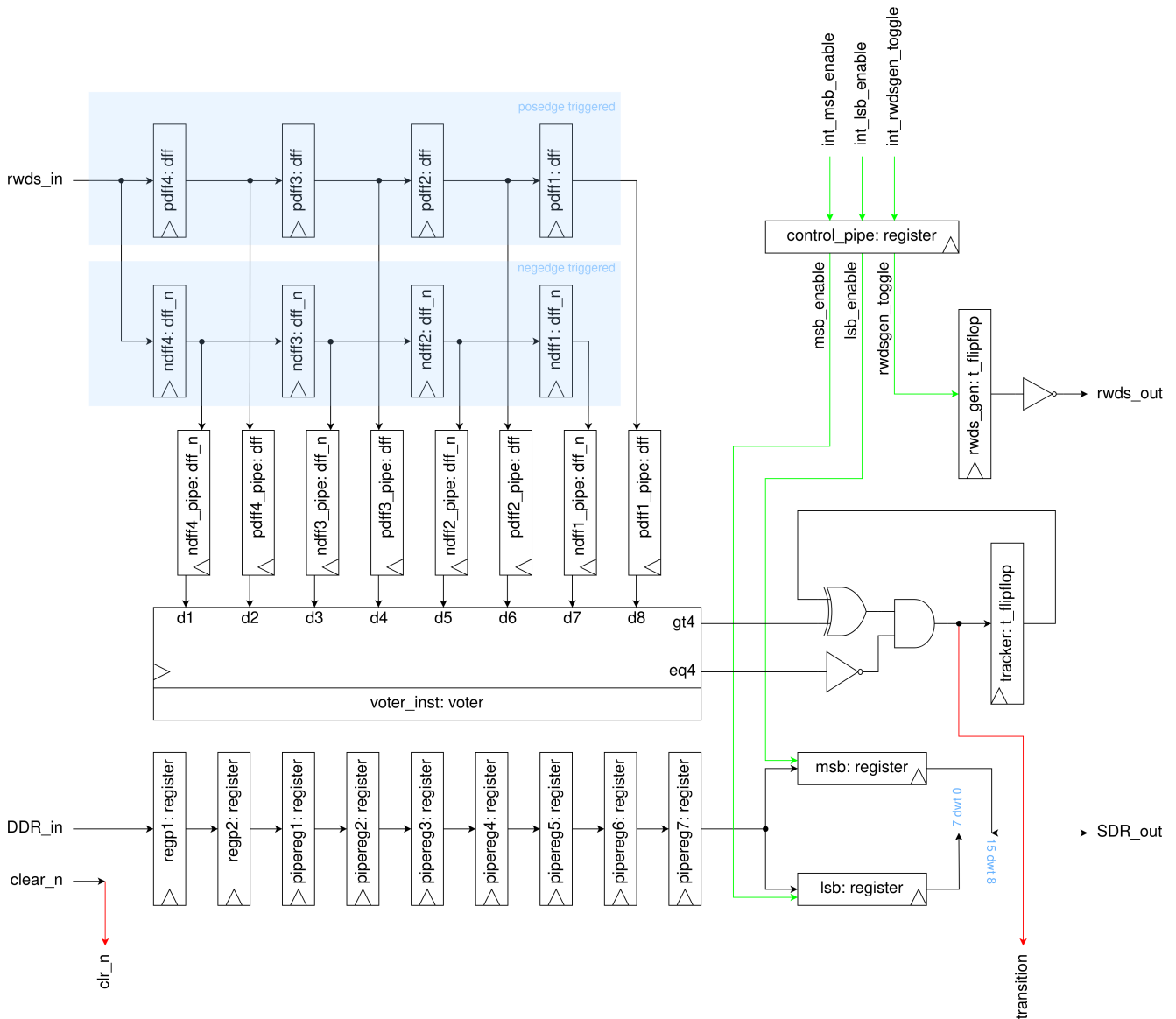


Figure 4.13: Execution unit of *readdata converter*. The top green signals are the control signals received from the control unit, whereas the bottom red signals are the status signals sent to the control unit.

4.1.2 Voter

The *voter* is able to compute the sum of its inputs and to notice if the result is equal (*eq4*) or greater (*gt4*) than four. As we can see in figure 4.14, it is divided in five different pipeline stages. The sum is computed by means of a set of half-adders and full-adders. Starting from the sum, *eq4* and *gt4* are generated using some logic gates.

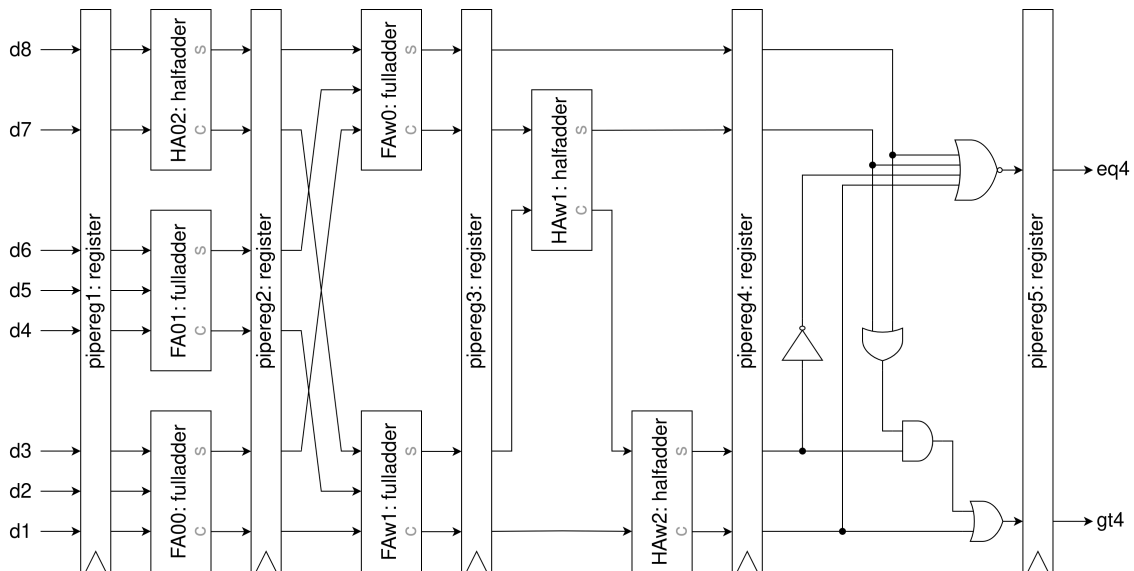


Figure 4.14: Architecture of the voter.

4.1.3 Control Unit

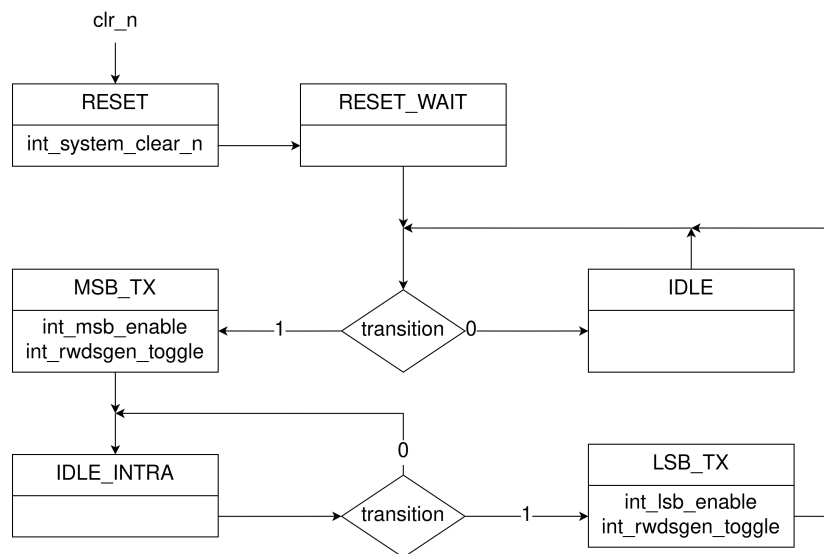


Figure 4.15: Control unit of readdata converter. It lets the data pass through the *lsb* register or the *msb* when a variation of decision is detected (refer to figure 4.1.1 for the execution unit). The register (*msb* or *lsb*) are selected alternatively to reconstruct a 16-bit SDR data.

4.2 Synchronizer

As described in section 4.1, readdata converter generates a 16-bit SDR output and a properly shifted strobe to sample it. The SDR output can be connected to the input data pin of a sampling register having the clock pin connected to the strobe, as shown in figure 4.9. However, in this way the data is not synchronized to the 50 MHz clock.

The main goal of the synchronizer, apart from providing the sampling register, is to synchronize the data with the 50 MHz clock. The main idea is to alternate multiple sampling registers, so that the sampled data remains available for a longer time. The architecture is divided in an execution unit (figure 4.17) and a control unit (figure 4.19).

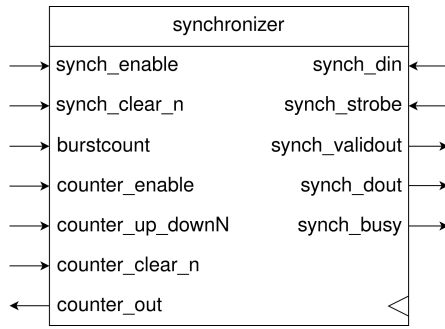


Figure 4.16: Top-level view of the *synchronizer*.

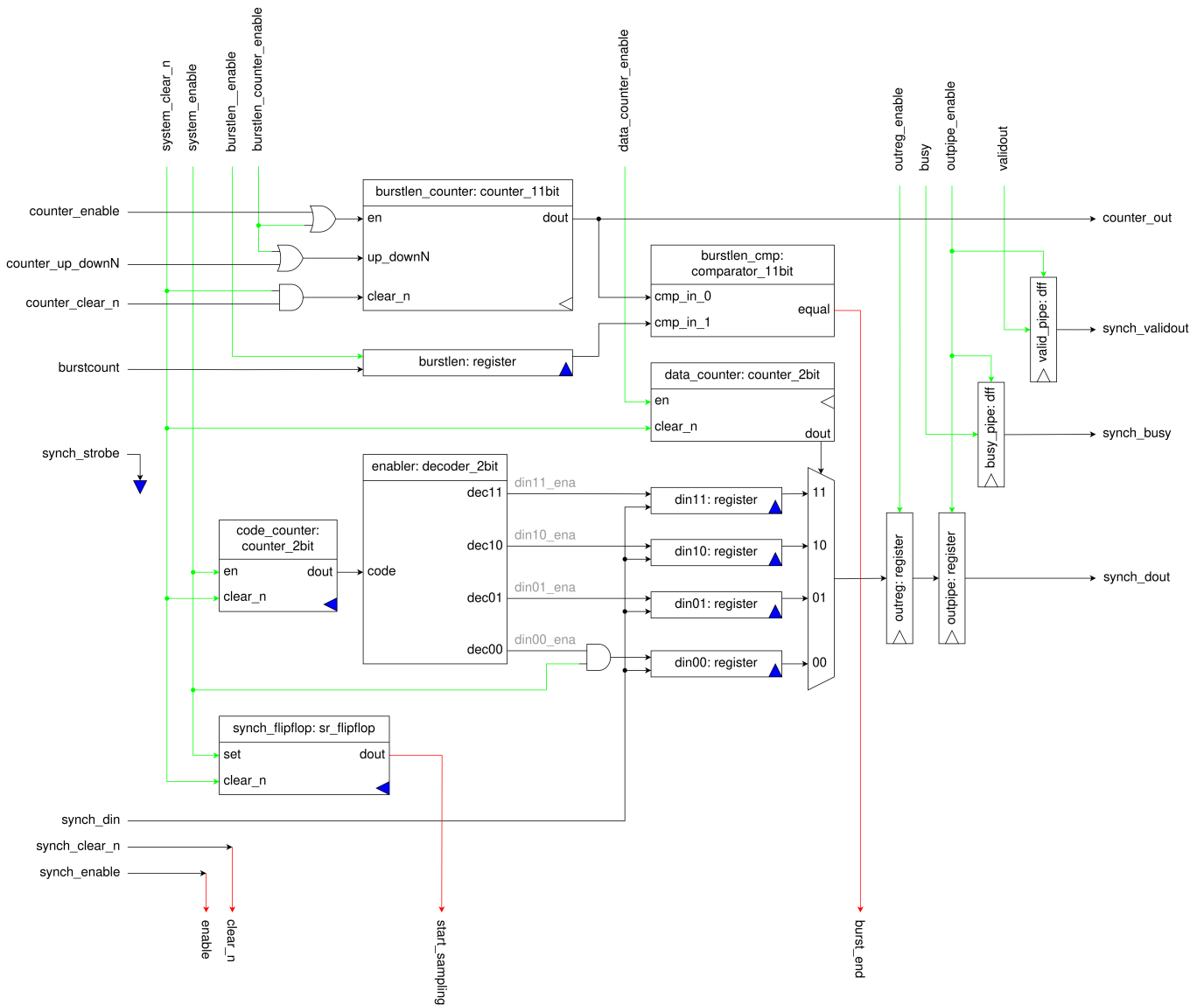


Figure 4.17: Execution unit of the *synchronizer*.

The actual synchronization is implemented by means of a set-reset flip-flop, namely the *synch_flipflop*, having the clock pin connected to the strobe, the set pin always set to logic 1 (assuming that the system is enabled) and the output pin connected to the FSM as a status signal called *start_sampling*. The strobe is asynchronous to the clock, therefore a variation of *start_sampling* may lead to setup/hold violations; in other words, the control unit may take more than one clock cycle to detect a variation of *start_sampling*. However, since the sampled data is available for

multiple clock cycles, this does not represent a problem. The timing diagram in figure 4.18 refers to a condition in which the activation of *start_sampling* is detected with a delay of 1 clock cycle.

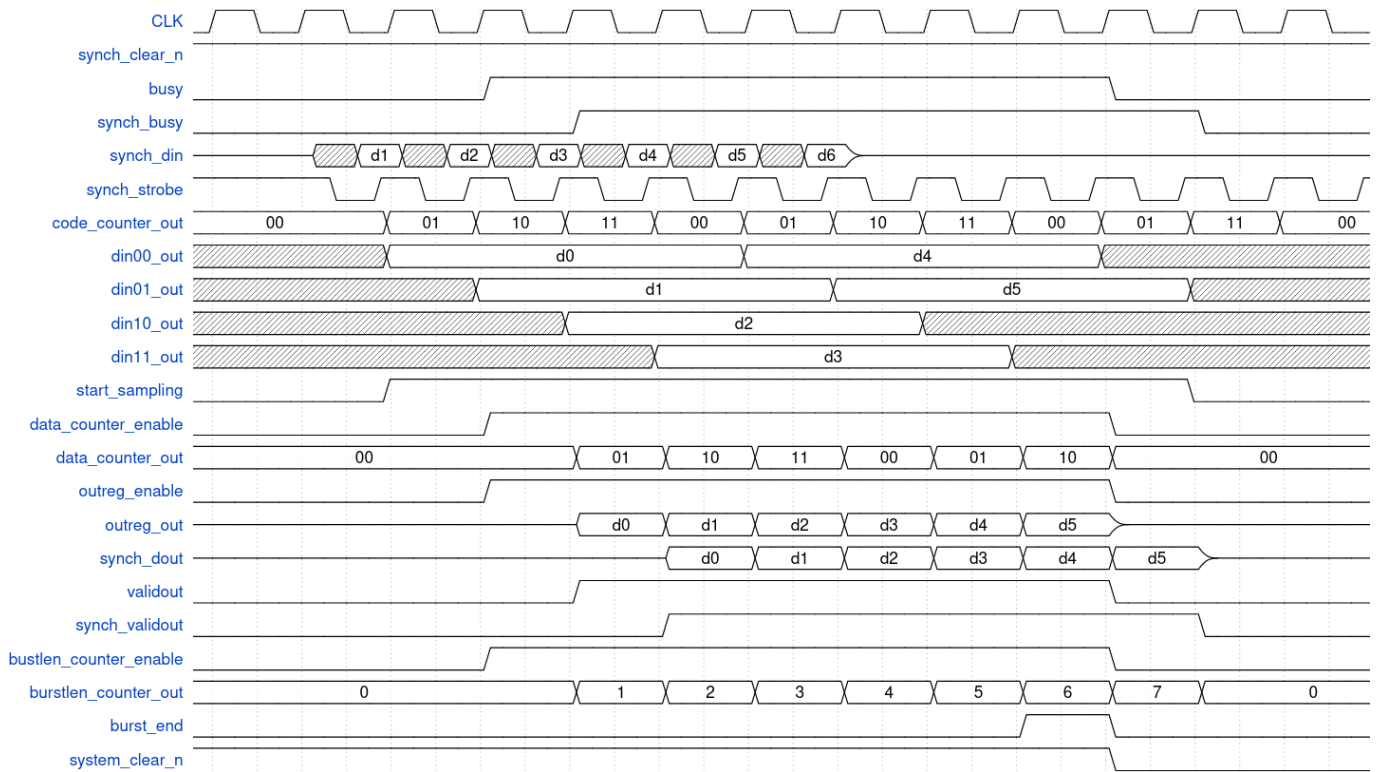


Figure 4.18: Timing diagram of the *synchronizer*.

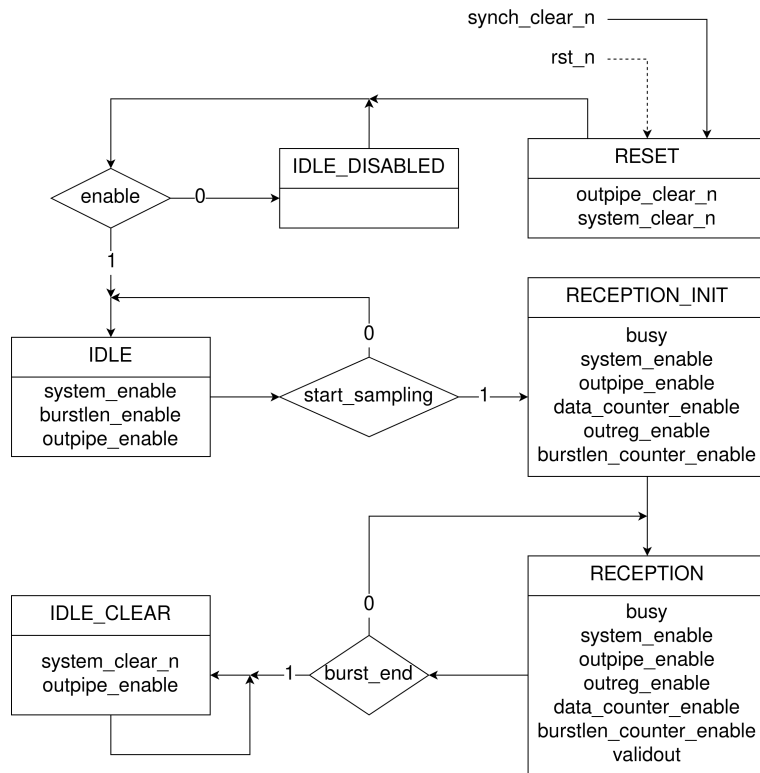


Figure 4.19: Control unit (finite-state machine) of the *synchronizer*.

The *synchronizer* is able to notice when the burst to be transferred is completed. The burst length is stored in a dedicated register called *burstlen*. The counter employed to keep track of the burst progress, namely the *burstlen_counter*, is made available outside, so that it can be employed independently when the *synchronizer* is not enabled.

Once the burst transfer is completed, the *synchronizer* remains in a state in which the FSM constantly reset all the components by means of the *system_clear_n* control signal, therefore it is not able to receive a new data burst. The activation of *system_clear_n* may be detected with a certain delay by the components that are clocked by the strobe, since it is synchronous to the clock. However, *system_clear_n* remains active for many clock cycles, therefore this does not represent a problem. To restart the *synchronizer* and make it able to receive a new burst, it is necessary to manually set the *synch_clear_n* signal after detecting *synch_busy* low, as shown in figure 4.18.

The *synchronizer* comes with an enable signal, namely *synch_enable*. As we can see in figure 4.19, this signal must be set before beginning a burst transfer (i.e. before the strobe starts oscillating), otherwise the strobe edges are ignored. Setting *synch_enable* after beginning a burst transfer causes an unknown behavior. If *synch_enable* is deactivated before completing a burst transfer, the transfer is completed anyway.

4.3 CA Builder

The 48-bit Command-Address is generated rearranging the bits of the address and combining them with the information about the operation type, as shown in figure 4.20.

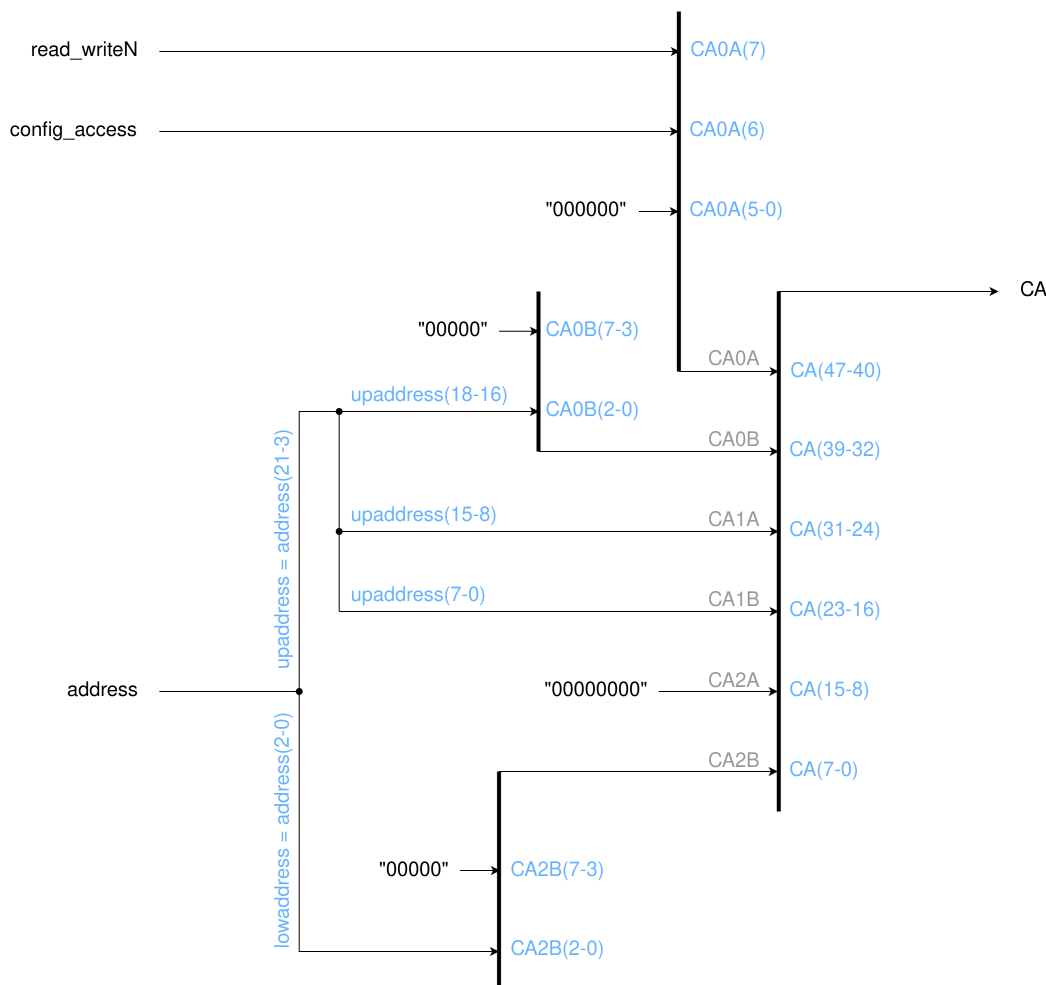


Figure 4.20: Architecture of the *CA builder*.

4.4 Writedata Converter

The Avalon side of the system works with a 16-bit SDR data, which must be converted in a 8-bit DDR data to be channeled in the HyperRAM DQ bus. This conversion is implemented in the *writedata converter*. A multiplexer having the selector connected to the system clock is exploited to alternate the more-significant byte and the less-significant byte of the SDR data, as represented in figure 4.21.

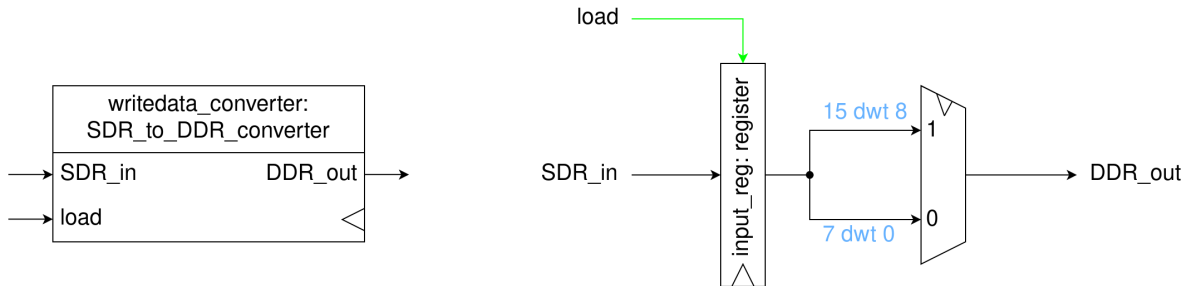


Figure 4.21: Top view (on the left) and internal architecture (on the right) of the *writedata converter*.

4.5 Configuration Builder

From the Avalon side, the host can only refer to the virtual configuration register. However, every time the virtual configuration register is written, the content of the physical configuration registers of the memory shall be updated. The *configuration builder* generates the content of the physical configuration registers starting from the content of the virtual configuration register, as shown in figure 4.22.

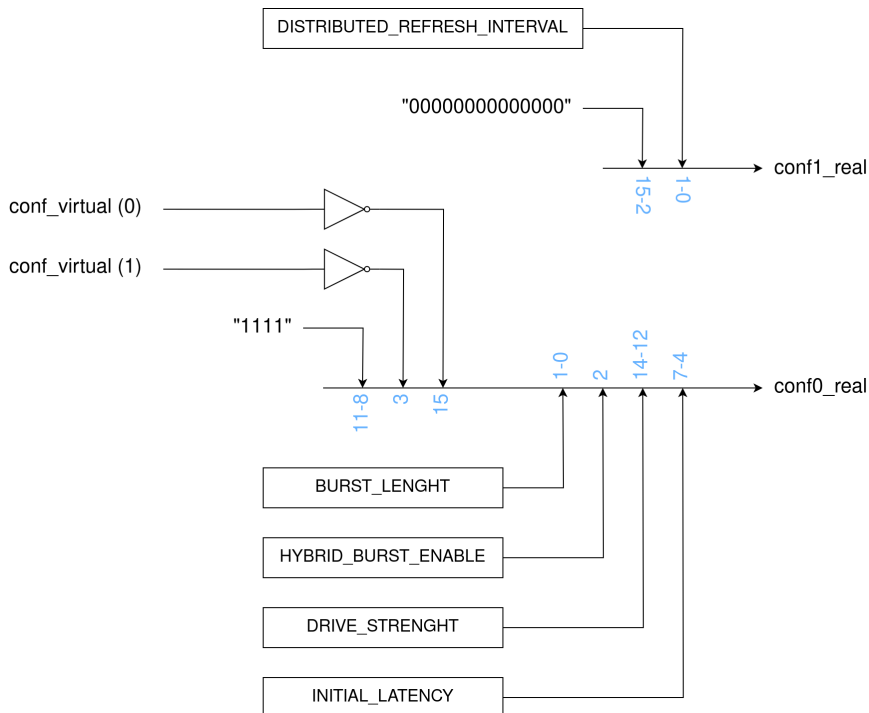


Figure 4.22: Architecture of the *configuration builder*.

4.6 CA Unpacker

The 48-bit Command-Address generated by the *CA builder* must be separated in three different 16-bit segments (namely *CA0*, *CA1* and *CA2*), so that it can be fed to the *writedata converter* and channeled in the HyperRAM DQ bus. This separation is implemented by the *CA unpacker*, represented in figure 4.23. The selection between *CA0*, *CA1* and *CA2* is achieved through a dedicated selector.

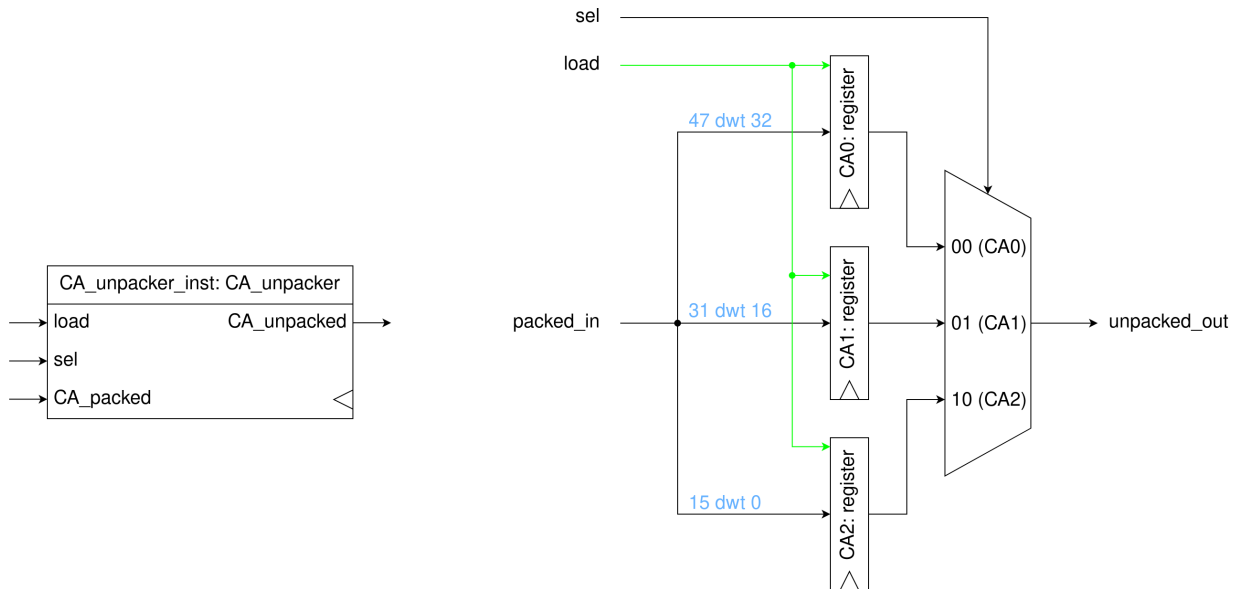


Figure 4.23: Top view (on the left) and internal architecture (on the right) of the *CA unpacker*.

4.7 Address Generator

When it comes to the Avalon communication protocol, the host can interrupt the burst transfer at any time during a write operation. However, the HyperRAM protocol does not support this feature. For this reason, the interface converter must terminate the operation toward the HyperRAM when the host interrupts the transfer and begin a new operation when the transfer is resumed. Consequently, any interruption of the transfer implies to reconstruct the start address (i.e. the address from which to start the operation) and to send again the command-address to the memory. For this reason, a burst interruption is really expensive in terms of latency and it would be better to avoid it.

During a write operation, the burst progress is tracked by means of the 11-bit counter provided by the *synchronizer* (*burstlen counter* referring to figure 4.17). When the host interrupts and resumes the transfer, the new start address can be generated from the output of this counter. In this regard, it is important to make some considerations:

- During a burst transfer the counter output is always two steps ahead with respect to the actual burst progress, so that the control unit can detect in advance when the burst is about to terminate (figure 4.24).
- When a burst transfer is interrupted (i.e. when the control unit switches from the *WRITEBURST* status to the *STOP_BURST_1* status) the counter out increases by one more step, although a new data has not been transferred (figure 4.24).
- The initial address provided by the host (namely the base address) remains stored in *addr_reg* (figure 4.3) until the transfer is completed.

All this considered, the start address can be reconstructed as $new\ start\ address = base\ address + counter\ out - 3$. Three pipeline registers are placed on the counter output (*cntpipe1*, *cntpipe2* and *cntpipe3* in figure 4.3) to obtain $counter\ out - 3$, whereas the sum with the base address is implemented using an adder (*addressgen* in figure 4.3).

4.8 Control Unit and Timing Diagrams

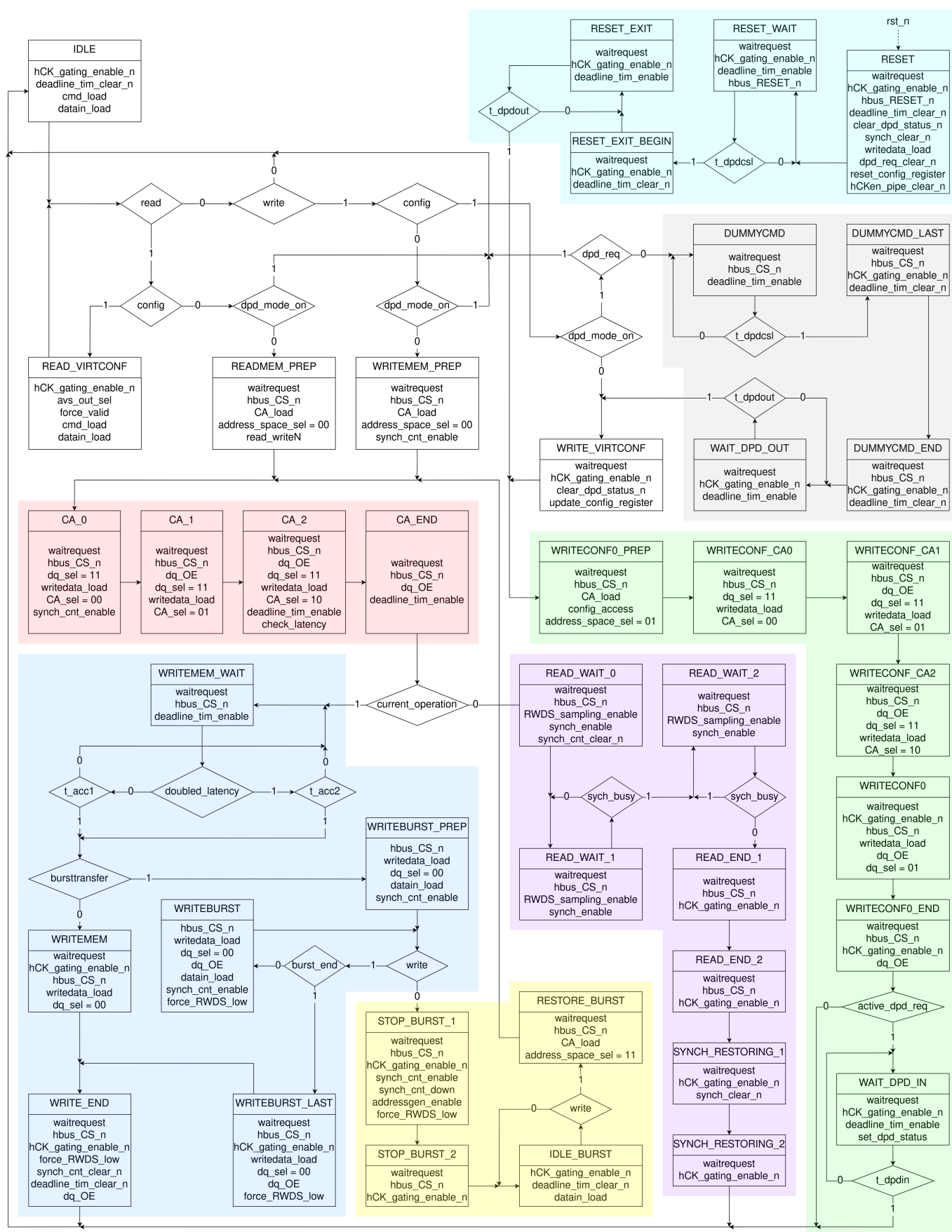


Figure 4.24: Control unit of the *avs.hram.mainconv* IP. Turquoise block: reset and power-on process. Grey block: DPD mode exit process. Green block: configuration registers update. Purple block: read operation. Blue block: write operation.

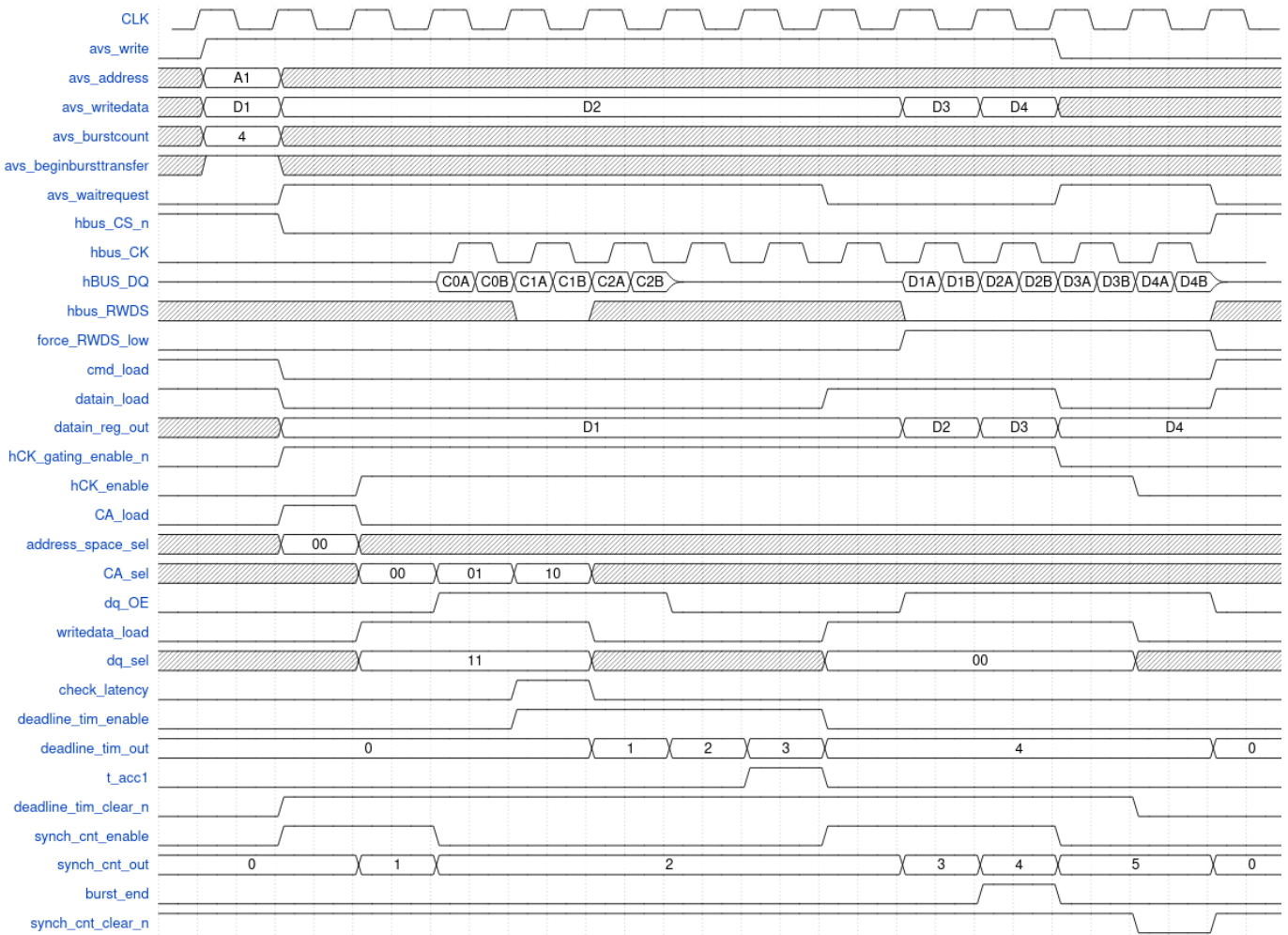


Figure 4.25: Write burst operation. The Avalon signals vary according to figure 2.2. The HyperRAM signals match the timing in figure 2.5.

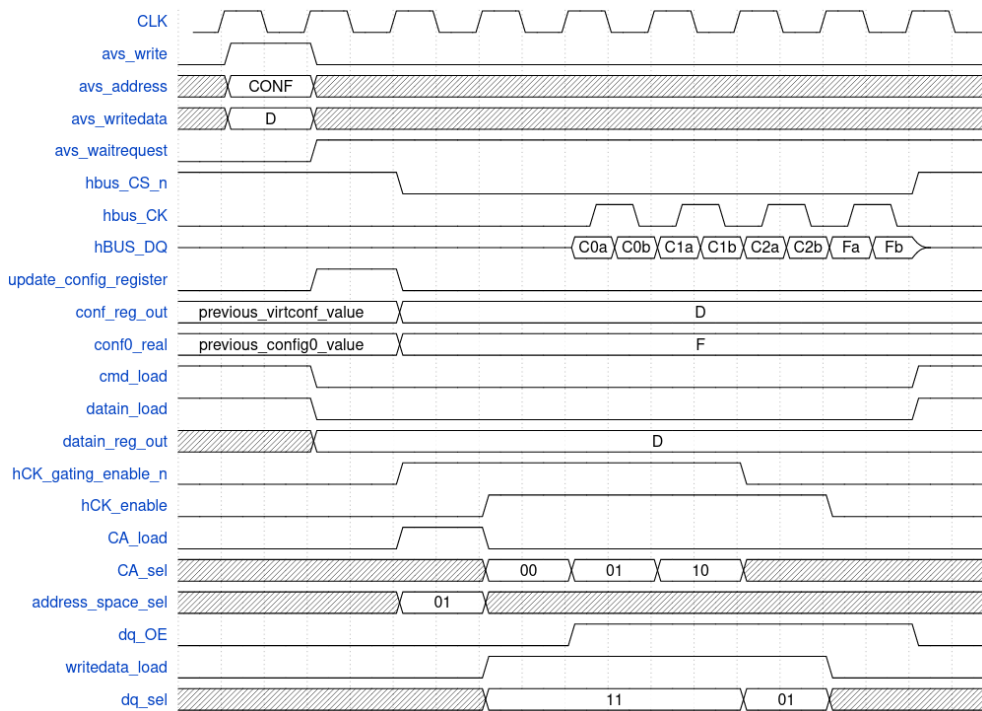


Figure 4.26: Configuration register update operation. The Avalon signals vary according to figure 2.2. The HyperRAM signals match the timing in figure 2.6.

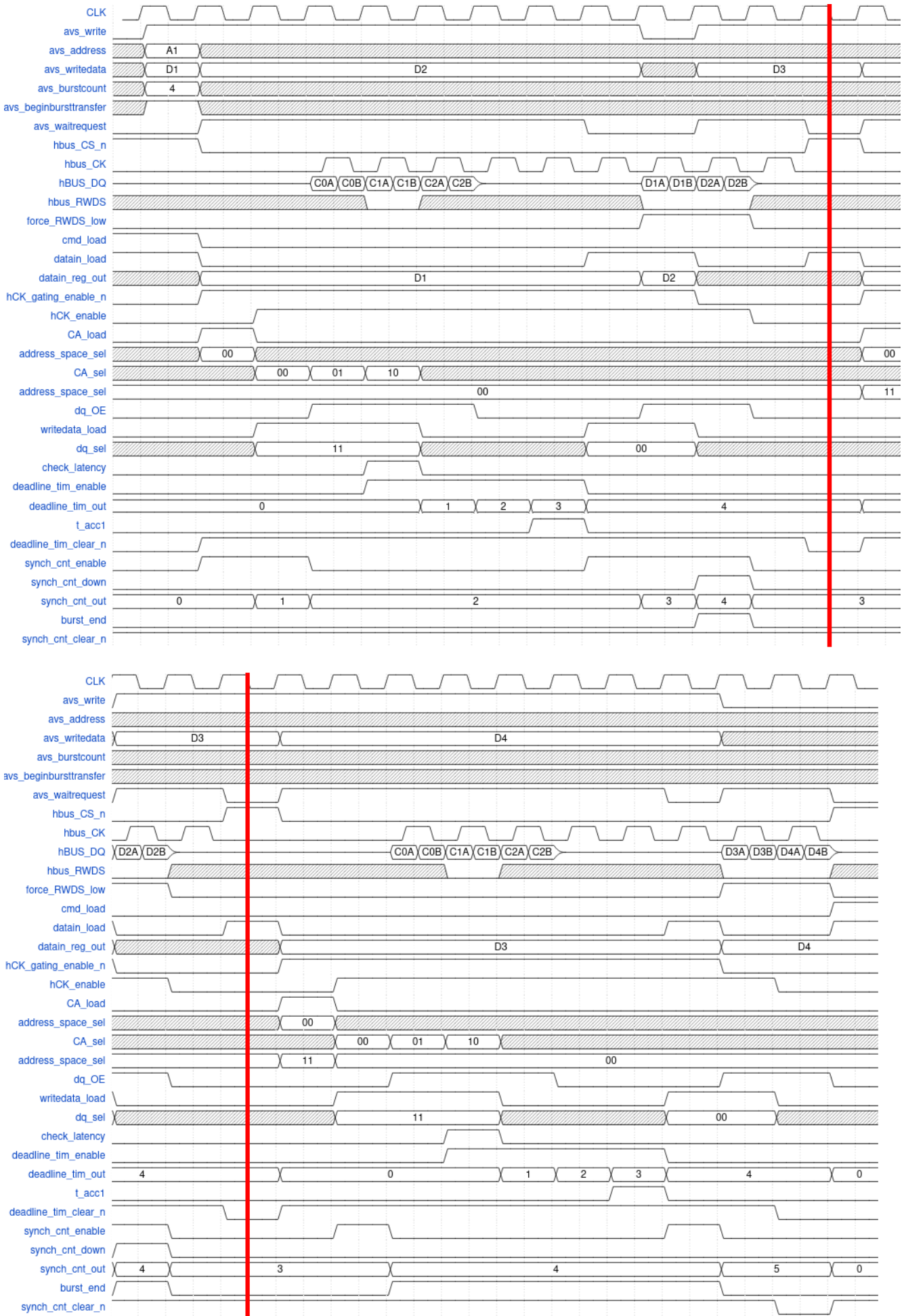


Figure 4.27: Write burst operation with burst interruption.

4.8.1 Memory Timing Constraints

As we can see in figures 2.4 and 2.5, several timing constraints (whose value is listed in figure 2.7) must be respected to ensure the correct functioning of the memory:

- T_{CSHI} : always ensured by the FSM
- T_{CSS} : always ensured by the FSM
- T_{RWR} : always ensured by the FSM
- T_{ACC} : always ensured by the FSM
- T_{CSH} : always ensured being null
- T_{DSV} : always ensured by the FSM
- T_{CKDS}, T_{CKD} : irrelevant, the trace delay is not considered
- T_{DSZ} (read operation) : irrelevant, only the edges of RWDS are significant
- T_{DSZ} (write operation) : irrelevant, RWDS is acquired early enough
- T_{DSS}, T_{DSH} : irrelevant, they do not affect the oversampling
- T_{OZ} : irrelevant, the value of DQ is relevant only in correspondence of the edges of RWDS
- T_{DQLZ} : irrelevant, the value of DQ is relevant only in correspondence of the edges of RWDS
- T_{CSM} : it limits the maximum burst length
- T_{IS}, T_{IH} : forced during synthesis

The constraints on T_{IS} and T_{IH} must be inserted in the SDC file related to the top level, i.e. the file in which the modular system containing the interface converter is instantiated.

T_{CSM} limits the maximum burst length. In the worst case (doubled access time, read operation), a latency of 12 clock periods must be considered between the assertion of $hram_CS_n$ and the beginning of the burst transfer (as we can see in figure 5.3). Moreover, at the end of the burst transfer 7 additional clock cycles are required before de-asserting $hram_CS_n$ (as shown in figure 5.4). As a result:

$$12 T_{CLK} + 7 T_{CLK} + N_{BURST} T_{CLK} < \max\{T_{CSM}\} = 4 \mu s \quad \Rightarrow \quad N_{BURST} < 181$$

As we can see, even though the burst length is expressed on 11 bits, only 9 bits are actually necessary.

TEST RESULTS

To test the interface converter against a large number of different stimuli it can be inserted in a processor-based system, as represented in figure 3.1. However, even if this approach allows to test the DUT under several different condition, it is possible that some particular operations of our interest are never tested. For instance, the processor may never try to put the memory in DPD mode. For this reason, it was decided to perform different analysis:

- **Preliminary simulation:** a custom testbench is created to directly drive the interface converter in order to verify its behavior under specific conditions.
- **Final simulation:** the interface converter is inserted in the processor-based system (figure 3.1) to simulate it against a large number of data.
- **Test on VirtLAB:** the processor-based system containing the interface converter is synthesized in the FPGA.

5.1 Preliminary Simulation

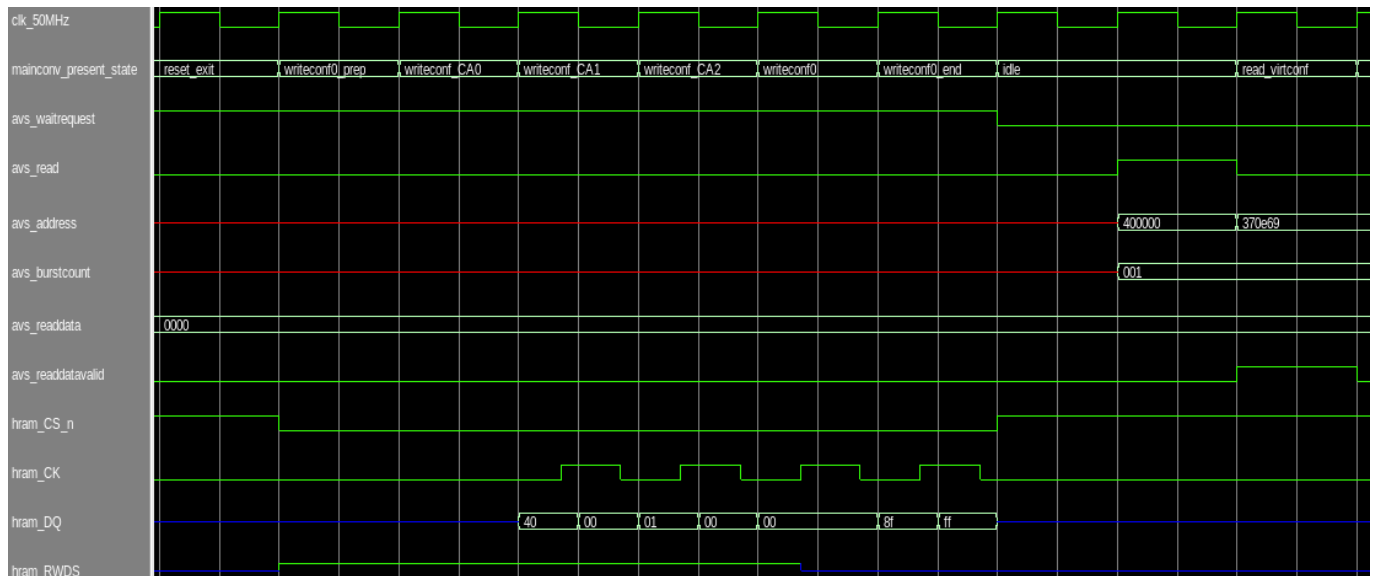


Figure 5.1: Automatic configuration register update after power-on.

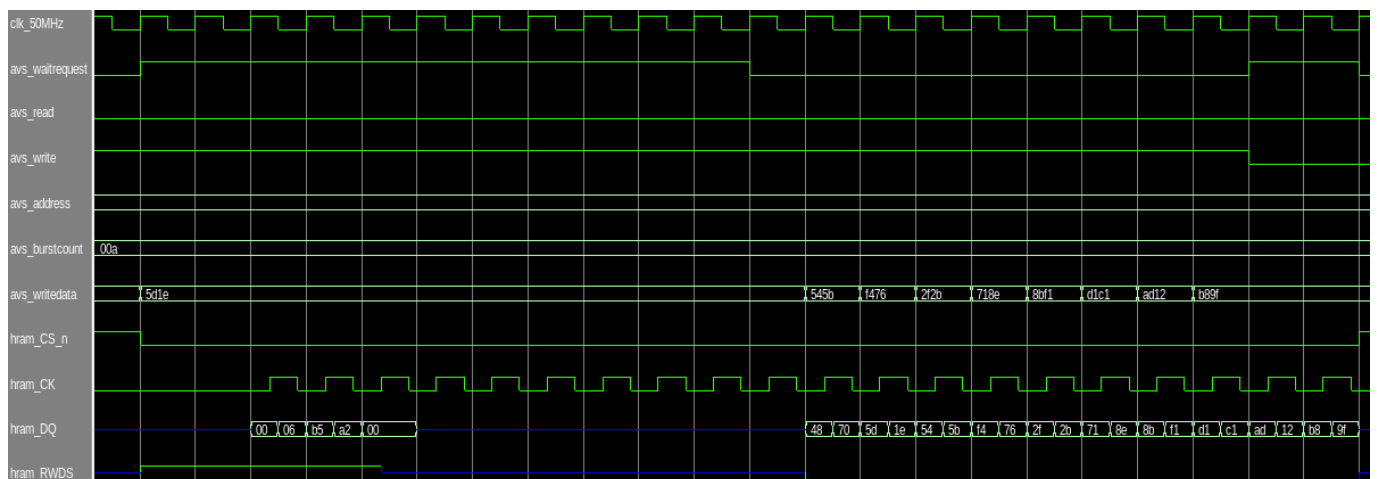


Figure 5.2: Write operation.

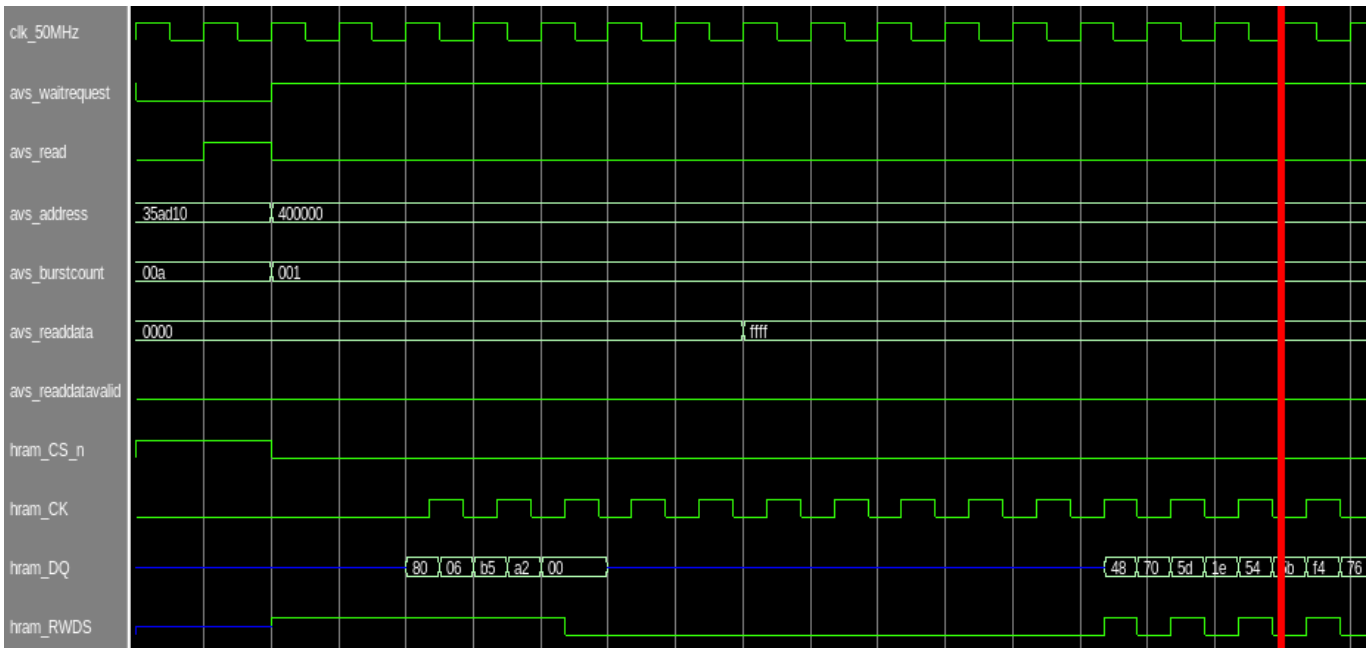


Figure 5.3: Read operation - part 1.

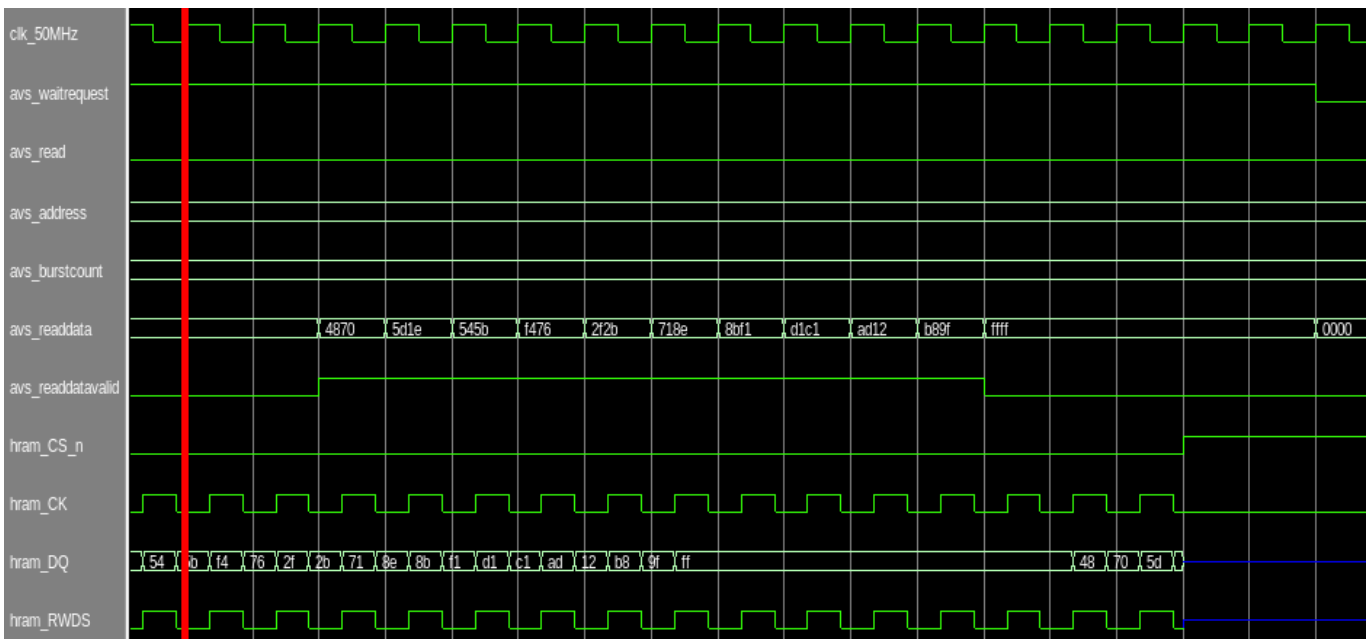


Figure 5.4: Read operation - part 2.

As we can see in figure 5.1, the system automatically initialize CR0 after the power-up as expected (refer to section 4.8). Moreover, a virtual configuration register read operation is successfully completed.

Other than the write operation (figure 5.2) and the read configuration (figures 5.3 and 5.4), it is important to simulate the behavior of the system in DPD mode. In particular, it must be able to:

- enter the DPD mode when requested (figure 5.5)
- ignore any operation beside a DPD exit request (figure 5.6)
- exit the DPD mode when requested (figures 5.6 and 5.7)
- automatically update the value of CR0 after exiting the DPD mode (figure 5.8)

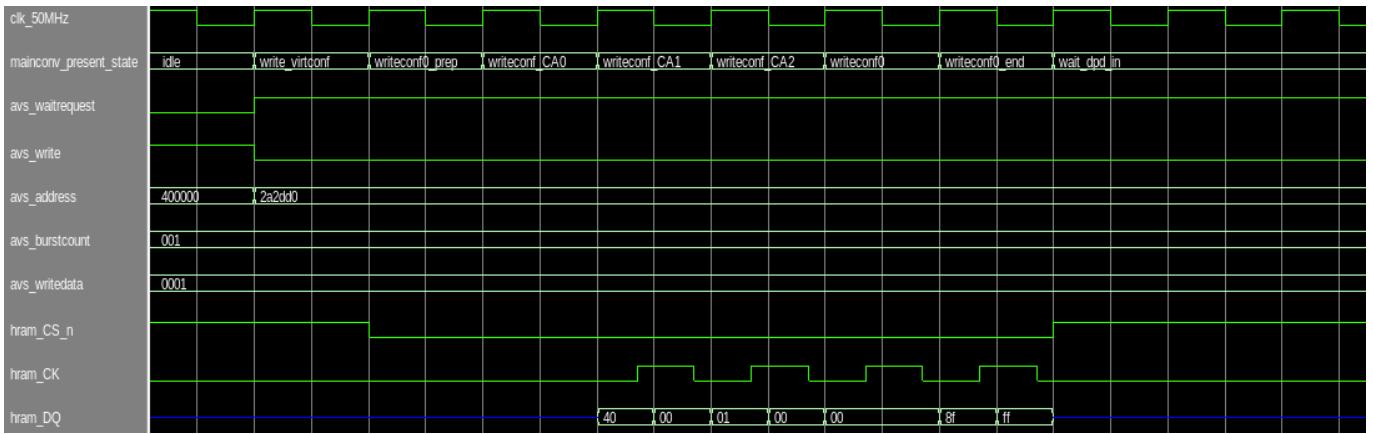


Figure 5.5: DPD mode entry request.

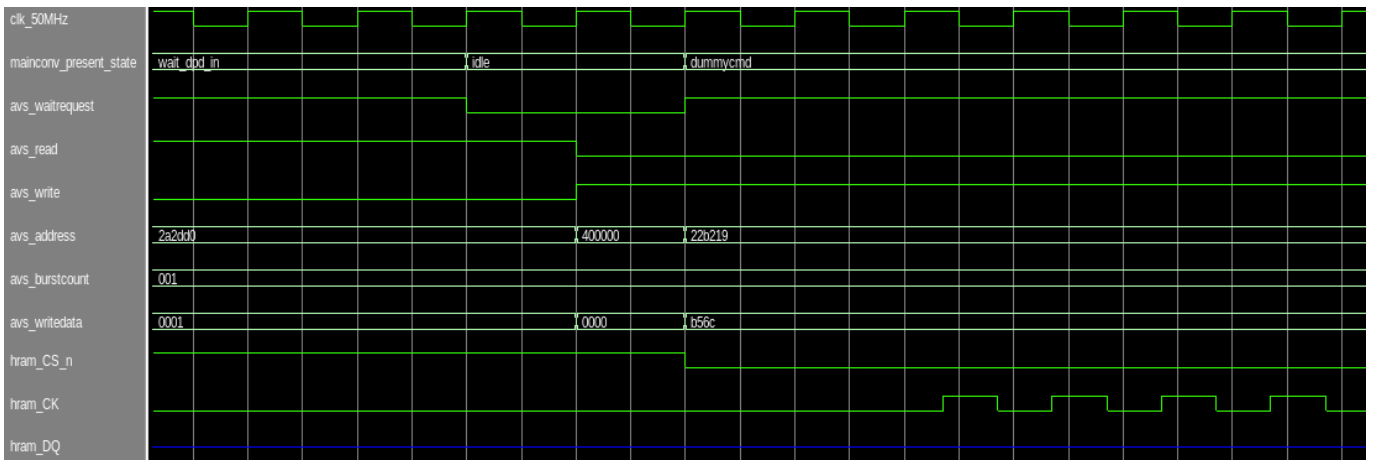


Figure 5.6: Ignored read operation and DPD mode exit request.

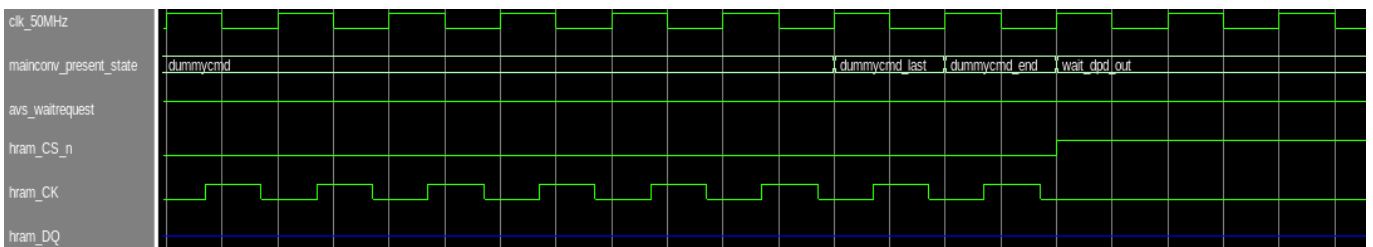


Figure 5.7: DPD mode exit.

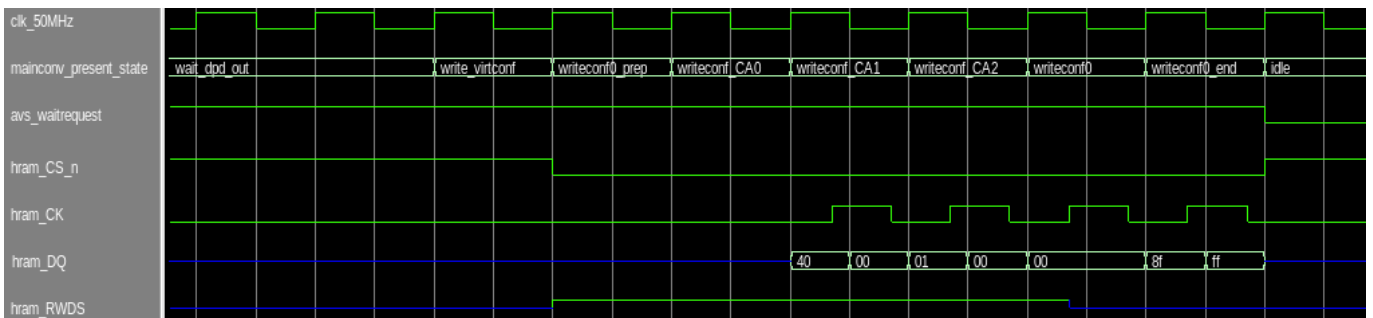


Figure 5.8: Automatic CR0 update after going back to normal mode.

5.2 Final Simulation

The interface converter is inserted in a processor-based system to simulate it against a large number of data, as we can see in figure 3.1. The *Nios II* processor is programmed to read four different signals provided by the *input generator* and to drive four different LEDs according to their value: when a status signal is set, the LED associated with it is turned on and vice-versa. The HyperRAM (together with the interface converter) is employed as data memory for the *Nios II* processor.

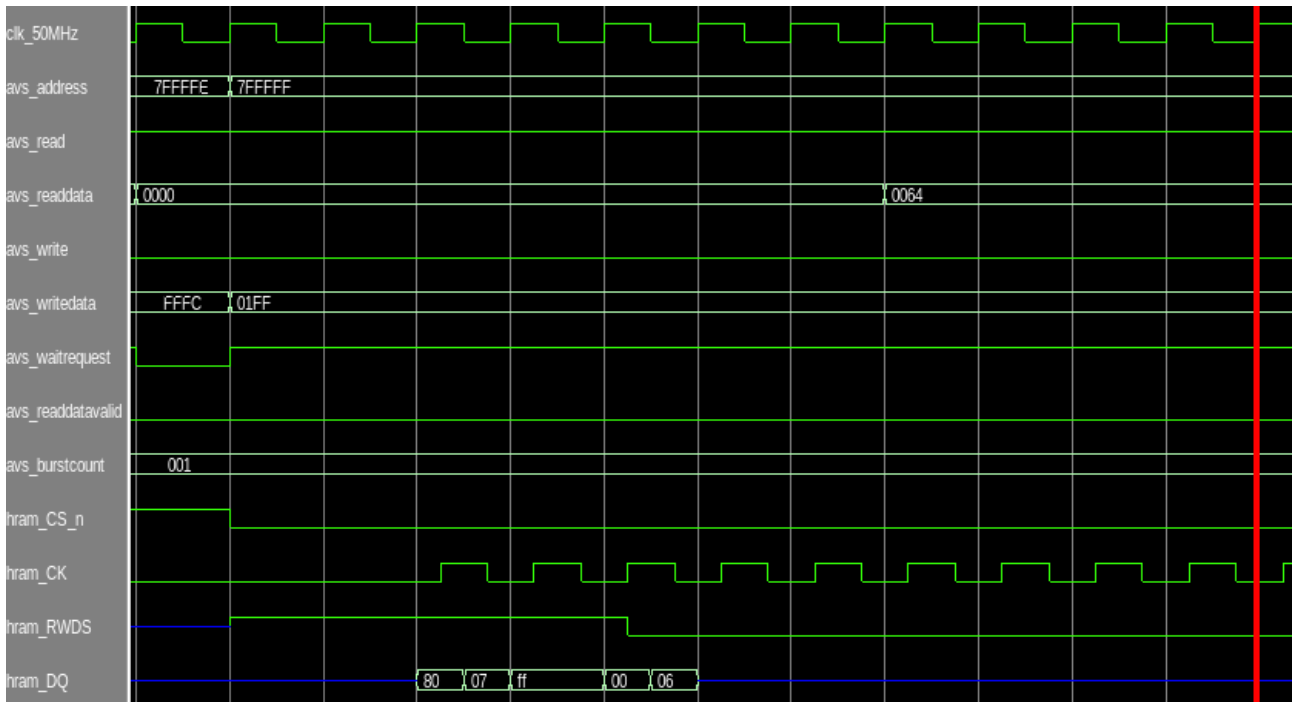


Figure 5.9: One of the many read operations driven by the processor - part 1.

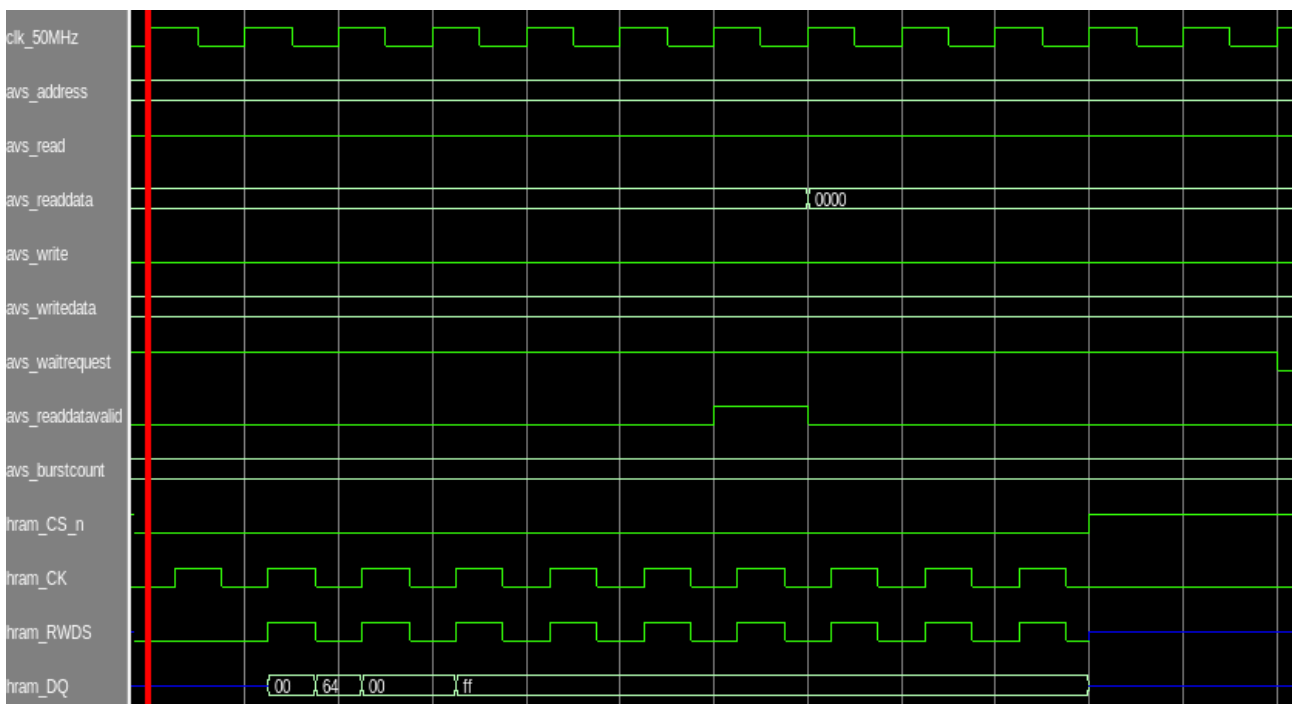


Figure 5.10: One of the many read operations driven by the processor - part 2.

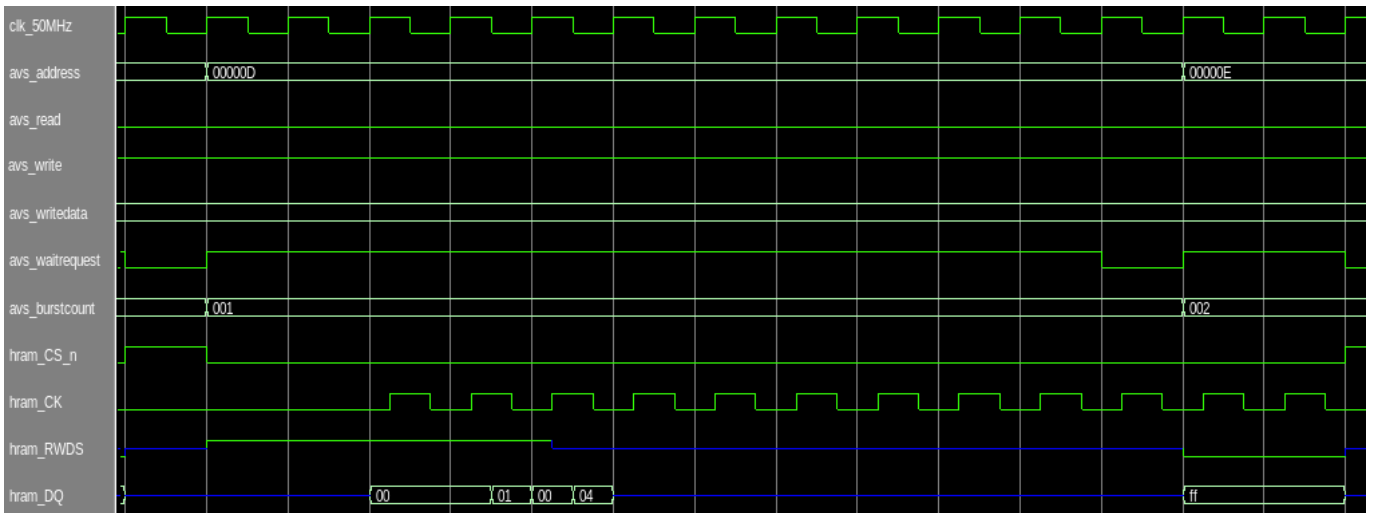


Figure 5.11: One of the many write operations driven by the processor.

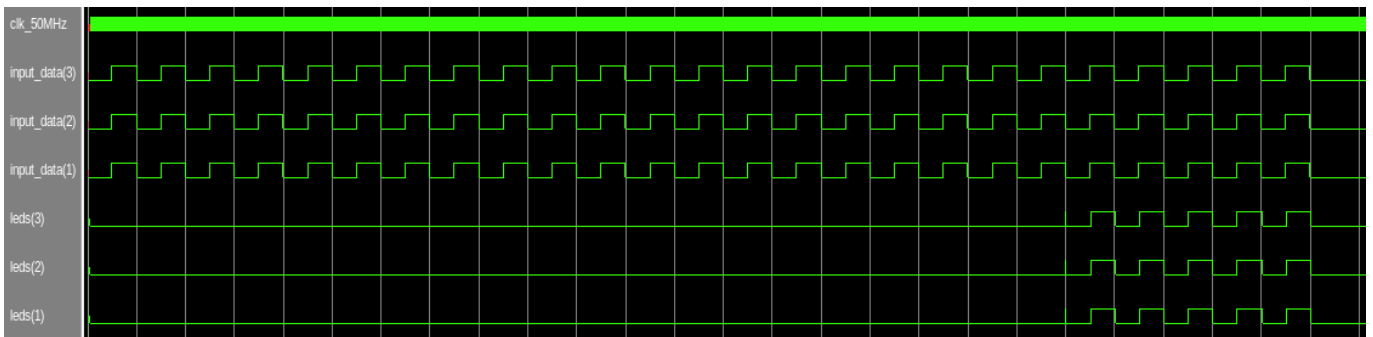


Figure 5.12: Processor-based system behavior.

At first, the processor-based system must wait for the memory to power-up, therefore it cannot process the input values, as we can see in figure 5.12. Indeed, during this initial time interval the LEDs are constantly turned off regardless of the value of the inputs. When the memory is powered-up, the processor drives the LEDs according to the input values as expected.

5.3 Test on VirtLAB

The processor-based system has been synthesized in the Cyclone 10 LP FPGA, verifying the correctness of its behavior. The *Nios II* processor has been programmed to read four different input signals and to drive for different LEDs according to their value. In particular, the input signals oscillates with a sufficiently low frequency, so that the consequent blinking of the LEDs is visible to the human eye.

FUTURE EXTENSIONS

The designed system is capable of correctly interface the HyperRAM with the Intel Avalon bus. However, it is still possible to work on some details to improve its performance. In particular:

- As described in section 4.1, the working frequency of the memory has been reduced to 50 MHz due to the limitations related to the oversampling. However, all the other components of the system are able to work at 100 MHz. Indeed, exploiting a controlled combinational delay (instead of the oversampling) to implement the *readdata converter* would allow to push the working frequency up to 100 MHz.
- On the Avalon side, the burst length is characterized by a parallelism of 11 bits. However, the burst length value cannot be higher than 181, therefore only 9 bits are actually useful, as described in section 4.8.1. For this reason, the counter and the comparator inside the *synchronizer* (*burstlen_counter* and *burstlen_cmp* referring to figure 4.17) can be implemented with a reduced parallelism, saving power and resources.
- The virtual configuration register allows to dynamically modify only two parameters of the HyperRAM (working mode and memory access latency). However, the design can be easily modified to allow the host to access also other parameters.

VHDL DESCRIPTION

Some of the components employed to realize the interface converter are implemented using already existing IP cores provided by the Intel catalog. Referring to figures 4.3, 4.2 and 4.17:

- *pll_x8*
- *dll_90*
- *counter_11bit_updown*
- *adder_22bit_1pipe*
- *adder_22bit_1pipe*
- *clkctrl*

All the other components require a custom HDL descriptions. The HDL descriptions of all the custom components are listed below (using VHDL as hardware description language).

```
1  -- BRIEF DESCRIPTION: flipflop type D
2
3  library    ieee;
4  use       ieee.std_logic_1164.all;
5  use       ieee.numeric_std.all;
6
7  entity d_flipflop is
8  port
9  (
10     clk       : in  std_logic;
11     enable    : in  std_logic;
12     clear_n   : in  std_logic;  -- synchronous clear, active low
13     reset_n   : in  std_logic;  -- asynchronous reset, active low
14     din       : in  std_logic;
15     dout      : out std_logic
16 );
17 end d_flipflop;
18
19 architecture behavior of d_flipflop is
20
21     begin
22
23         -- main process -----
24         dff_process: process (clk, clear_n, reset_n, enable)
25         begin
26             if (reset_n = '0') then
27                 dout <= '0';
28             else
29                 if (rising_edge(clk)) then
30                     if (clear_n = '0') then
31                         dout <= '0';
32                     elsif (enable = '1') then
33                         dout <= din;
34                     end if;
35                 end if;
36             end if;
37         end process dff_process; -----
38
39     end behavior;
```

```
1  -- BRIEF DESCRIPTION: flipflop type D
2
3  library    ieee;
4  use       ieee.std_logic_1164.all;
```

```

5 use      ieee.numeric_std.all;
6
7 entity dff_negedge is
8 port
9 (
10  clk       : in  std_logic;
11  enable    : in  std_logic;
12  clear_n   : in  std_logic; -- synchronous clear, active low
13  reset_n   : in  std_logic; -- asynchronous reset, active low
14  din       : in  std_logic;
15  dout      : out std_logic
16 );
17 end dff_negedge;
18
19 architecture behavior of dff_negedge is
20
21 begin
22
23     -- main process -----
24     dff_process: process (clk, clear_n, reset_n, enable)
25     begin
26         if (reset_n = '0') then
27             dout <= '0';
28         else
29             if (falling_edge(clk)) then
30                 if (clear_n = '0') then
31                     dout <= '0';
32                 elsif (enable = '1') then
33                     dout <= din;
34                 end if;
35             end if;
36         end if;
37     end process dff_process; -----
38
39 end behavior;

```

```

1 -- BRIEF DESCRIPTION: flip flop type T (toggle)
2 -- the output toggles at each clock cycle
3
4 library  ieee;
5 use      ieee.std_logic_1164.all;
6 use      ieee.numeric_std.all;
7
8 entity t_flipflop is
9 port
10 (
11  clk       : in  std_logic;
12  enable    : in  std_logic;
13  clear_n   : in  std_logic; -- synchronous clear, active low
14  reset_n   : in  std_logic; -- asynchronous reset, active low
15  din       : in  std_logic;
16  dout      : out std_logic := '0'
17 );
18 end t_flipflop;
19
20
21 architecture behavior of t_flipflop is
22
23     signal dummy_out: std_logic;
24
25 begin
26
27     -- main process -----
28     tff_process: process (clk, clear_n, reset_n, enable)
29     begin
30         if (reset_n = '0') then
31             dummy_out <= '0';
32         else
33             if (rising_edge(clk)) then
34                 if (clear_n = '0') then
35                     dummy_out <= '0';
36                 elsif (enable = '1') then
37                     if (din = '0') then
38                         dummy_out <= dummy_out;

```



```

39         elsif (din = '1') then
40             dummy_out <= not dummy_out;
41         end if;
42     end if;
43 end if;
44 end if;
45 end process tff_process; -----
46
47 dout <= dummy_out;
48
49 end behavior;

```

```

1  -- BRIEF DESCRIPTION: set-reset flipflop
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5
6  entity sr_flipflop is
7  port
8  (
9      clk      : in std_logic;
10     set      : in std_logic;
11     clear_n  : in std_logic;
12     rst_n    : in std_logic;
13     dout     : out std_logic
14 );
15 end sr_flipflop;
16
17 architecture behavior of sr_flipflop is
18
19     begin
20
21         -- main process -----
22         sr_process: process (clk, clear_n, set, rst_n)
23         begin
24             if (rst_n = '0') then
25                 dout <= '0';
26             else
27                 if (rising_edge(clk)) then
28                     if (clear_n = '0') then
29                         dout <= '0';
30                     elsif (set = '1') then
31                         dout <= '1';
32                     end if;
33                 end if;
34             end if;
35         end process sr_process; -----
36
37     end behavior;

```

```

1  -- BRIEF DESCRIPTION: N-bit register
2
3  library     ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity reg is
8  generic
9  (
10     N : integer := 8
11 );
12 port
13 (
14     clk      : in  std_logic;
15     enable   : in  std_logic;
16     clear_n  : in  std_logic;  -- synchronous clear, active low
17     reset_n  : in  std_logic;  -- asynchronous clear, active low
18     din      : in  std_logic_vector(N-1 downto 0);
19     dout     : out std_logic_vector(N-1 downto 0) := (others => '0')
20 );

```

```

21 end reg;
22
23 architecture behavior of reg is
24
25     begin
26
27         -- main process -----
28         reg_process: process (clk, clear_n, reset_n, enable, din)
29         begin
30             if (reset_n = '0') then
31                 dout <= (others => '0');
32             else
33                 if (rising_edge(clk)) then
34                     if (clear_n = '0') then
35                         dout <= (others => '0');
36                     elsif (enable = '1') then
37                         dout <= din;
38                     end if;
39                 end if;
40             end if;
41         end process reg_process; -----
42
43 end behavior;

```

```

1  -- BRIEF DESCRIPTION: N-bit register, negative-edge triggered
2
3  library    ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity reg_negedge is
8  generic
9  (
10     N : integer := 8
11 );
12 port
13 (
14     clk      : in  std_logic;
15     enable   : in  std_logic;
16     clear_n  : in  std_logic;  -- synchronous clear, active low
17     reset_n  : in  std_logic;  -- asynchronous clear, active low
18     din      : in  std_logic_vector(N-1 downto 0);
19     dout     : out std_logic_vector(N-1 downto 0) := (others => '0')
20 );
21 end reg_negedge;
22
23 architecture behavior of reg_negedge is
24
25     begin
26
27         -- main process -----
28         reg_process: process (clk, clear_n, reset_n, enable, din)
29         begin
30             if (reset_n = '0') then
31                 dout <= (others => '0');
32             else
33                 if (falling_edge(clk)) then
34                     if (clear_n = '0') then
35                         dout <= (others => '0');
36                     elsif (enable = '1') then
37                         dout <= din;
38                     end if;
39                 end if;
40             end if;
41         end process reg_process; -----
42
43 end behavior;

```

```

1  -- BRIEF DESCRIPTION: multiplexer 2-inputs 1-output N-bit
2

```

```

3 library    ieee;
4 use        ieee.std_logic_1164.all;
5 use        ieee.numeric_std.all;
6
7 entity mux_2to1 is
8 generic
9 (
10  N : integer := 1
11 );
12 port
13 (
14  din_0    : in   std_logic_vector((N-1) downto 0);
15  din_1    : in   std_logic_vector((N-1) downto 0);
16  sel      : in   std_logic;
17  dout     : out  std_logic_vector((N-1) downto 0)
18 );
19 end mux_2to1;
20
21 architecture behavior of mux_2to1 is
22
23 begin
24
25     -- main process -----
26     mux_output_evaluation: process (sel, din_0, din_1)
27     begin
28         case sel is
29             when '0' =>
30                 dout <= din_0;
31             when others =>
32                 dout <= din_1;
33         end case;
34     end process mux_output_evaluation; -----
35
36 end behavior;

```

```

1 -- BRIEF DESCRIPTION: multiplexer 4-inputs 1-output N-bit
2
3 library    ieee;
4 use        ieee.std_logic_1164.all;
5 use        ieee.numeric_std.all;
6
7 entity mux_4to1 is
8 generic
9 (
10  N : integer := 1
11 );
12 port
13 (
14  din_00   : in   std_logic_vector((N-1) downto 0);
15  din_01   : in   std_logic_vector((N-1) downto 0);
16  din_10   : in   std_logic_vector((N-1) downto 0);
17  din_11   : in   std_logic_vector((N-1) downto 0);
18  sel      : in   std_logic_vector(1 downto 0);
19  dout     : out  std_logic_vector((N-1) downto 0)
20 );
21 end mux_4to1;
22
23 architecture behavior of mux_4to1 is
24
25 begin
26
27     -- main process -----
28     output_evaluation: process (sel, din_00, din_01, din_10, din_11)
29     begin
30         case sel is
31             when "00" =>
32                 dout <= din_00;
33             when "01" =>
34                 dout <= din_01;
35             when "10" =>
36                 dout <= din_10;
37             when others =>
38                 dout <= din_11;
39         end case;

```

```

40     end process output_evaluation; -----
41
42 end behavior;

```

```

1  -- BRIEF DESCRIPTION: non-inverting tristate buffer
2
3  library    ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity tristate_buffer is
8  generic
9  (
10     N : integer := 8
11 );
12 port
13 (
14     enable    : in  std_logic;
15     din       : in  std_logic_vector(N-1 downto 0);
16     dout      : out std_logic_vector(N-1 downto 0)
17 );
18 end tristate_buffer;
19
20 architecture behavior of tristate_buffer is
21
22     begin
23
24         dout <= din when enable = '1' else (others => 'Z');
25
26 end behavior;

```

```

1  -- BRIEF DESCRIPTION: comparison between two inputs (N bit)
2  -- "equal" is set if the two inputs are the same
3
4  library    ieee;
5  use        ieee.std_logic_1164.all;
6  use        ieee.numeric_std.all;
7
8  entity comparator_Nbit is
9  generic
10 (
11     N : integer := 1
12 );
13 port
14 (
15     din_0    : in    std_logic_vector((N-1) downto 0);
16     din_1    : in    std_logic_vector((N-1) downto 0);
17     equal    : out   std_logic
18 );
19 end comparator_Nbit;
20
21 architecture behavior of comparator_Nbit is
22
23     signal bitwise_xnor : std_logic_vector(N-1 downto 0);
24     signal intra_and    : std_logic_vector(N downto 0);
25
26     begin
27
28         g1: for i in 0 to N-1 generate
29             bitwise_xnor(i) <= din_0(i) xnor din_1(i);
30         end generate;
31
32         intra_and(0) <= bitwise_xnor(0);
33
34         g2: for j in 1 to N generate
35             intra_and(j) <= intra_and(j-1) and bitwise_xnor(j-1);
36         end generate;
37
38         equal <= intra_and(N);
39
40 end behavior;

```

```

1  -- BRIEF DESCRIPTION: synchronous counter N bit
2  -- it can count up to (2^N)-1
3  -- it is implemented as a chain of T flipflops
4
5  library    ieee;
6  use        ieee.std_logic_1164.all;
7  use        ieee.numeric_std.all;
8
9  entity counter_Nbit is
10 generic
11 (
12     N : integer := 4
13 );
14 port
15 (
16     clk       : in  std_logic;
17     enable    : in  std_logic;
18     clear_n   : in  std_logic;
19     reset_n   : in  std_logic;
20     dout      : out std_logic_vector(N-1 downto 0)
21 );
22 end counter_Nbit;
23
24 architecture rtl of counter_Nbit is
25
26     -- COMPONENT: flip flop type T -----
27     component t_flipflop is
28         port
29         (
30             clk       : in  std_logic;
31             enable    : in  std_logic;
32             clear_n   : in  std_logic; -- synchronous clear, active low
33             reset_n   : in  std_logic; -- asynchronous reset, active low
34             din       : in  std_logic;
35             dout      : out std_logic := '0'
36         );
37     end component; -----
38
39     -- SIGNALS -----
40     signal tff_in   : std_logic_vector(N-1 downto 0);
41     signal tff_out  : std_logic_vector(N-1 downto 0);
42     -----
43
44     begin
45
46         tff_in(0) <= enable;
47
48         -- first flip flop of the chain -----
49         entry_tff: t_flipflop
50         port map
51         (
52             clk       => clk,
53             enable    => '1',
54             clear_n   => clear_n,
55             reset_n   => reset_n,
56             din       => tff_in(0),
57             dout      => tff_out(0)
58         ); -----
59
60         -- chain generation -----
61         g1: for i in 1 to N-1 generate
62             tff_in(i) <= tff_in(i-1) and tff_out(i-1);
63             chain_tff: t_flipflop
64             port map
65             (
66                 clk       => clk,
67                 enable    => '1',
68                 clear_n   => clear_n,
69                 reset_n   => reset_n,
70                 din       => tff_in(i),
71                 dout      => tff_out(i)
72             );
73         end generate; -----
74
75         dout <= tff_out;
76

```

```

77 end rtl;

1  -- BRIEF DESCRIPTION: 14-bit timer with alert on 3, 7, 20, 1000 and 15000
2
3  library    ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity timer_14bit is
8  port
9  (
10     clk          : in  std_logic;
11     enable       : in  std_logic;
12     clear_n      : in  std_logic;
13     tim_3        : out std_logic;
14     tim_7        : out std_logic;
15     tim_21       : out std_logic;
16     tim_1000    : out std_logic;
17     tim_15000   : out std_logic
18 );
19 end timer_14bit;
20
21 architecture rtl of timer_14bit is
22
23     -- COMPONENT: counter N-bit
24     -----
25     component counter_Nbit is
26     generic
27     (
28         N : integer := 4
29     );
30     port
31     (
32         clk          : in  std_logic;
33         enable       : in  std_logic;
34         clear_n      : in  std_logic;
35         reset_n      : in  std_logic;
36         dout         : out std_logic_vector(N-1 downto 0)
37     );
38     end component; -----
39
40     signal cnt_out: std_logic_vector(13 downto 0);
41
42     begin
43
44         -- counter 14-bit -----
45         cnt: counter_Nbit
46         generic map
47         (
48             N => 14
49         )
50         port map
51         (
52             clk          => clk,
53             enable       => enable,
54             clear_n      => clear_n,
55             reset_n      => '1',
56             dout         => cnt_out
57         ); -----
58
59         -- alert on dout = 3 -----
60         tim_3    <= (cnt_out(0))    and (cnt_out(1))    and (not cnt_out(2)) and
61                   (not cnt_out(3)) and (not cnt_out(4)) and (not cnt_out(5)) and
62                   (not cnt_out(6)) and (not cnt_out(7)) and (not cnt_out(8)) and
63                   (not cnt_out(9)) and (not cnt_out(10)) and (not cnt_out(11)) and
64                   (not cnt_out(12)) and (not cnt_out(13)) ;
65
66         -- alert on dout = 7 -----
67         tim_7    <= (cnt_out(0))    and (cnt_out(1))    and (cnt_out(2))    and
68                   (not cnt_out(3)) and (not cnt_out(4)) and (not cnt_out(5)) and
69                   (not cnt_out(6)) and (not cnt_out(7)) and (not cnt_out(8)) and
70                   (not cnt_out(9)) and (not cnt_out(10)) and (not cnt_out(11)) and
71                   (not cnt_out(12)) and (not cnt_out(13));

```

```

72 -----
73
74 -- alert on dout = 20 -----
75 tim_21    <= (cnt_out(0))    and (not cnt_out(1)) and (cnt_out(2))    and
76             (not cnt_out(3)) and (cnt_out(4))    and (not cnt_out(5)) and
77             (not cnt_out(6)) and (not cnt_out(7)) and (not cnt_out(8)) and
78             (not cnt_out(9)) and (not cnt_out(10)) and (not cnt_out(11)) and
79             (not cnt_out(12)) and (not cnt_out(13));
80 -----
81
82 -- alert on dout = 1000 -----
83 tim_1000  <= (not cnt_out(0)) and (not cnt_out(1)) and (not cnt_out(2)) and
84             (cnt_out(3))    and (not cnt_out(4)) and (cnt_out(5))    and
85             (cnt_out(6))    and (cnt_out(7))    and (cnt_out(8))    and
86             (cnt_out(9))    and (not cnt_out(10)) and (not cnt_out(11)) and
87             (not cnt_out(12)) and (not cnt_out(13));
88 -----
89
90 -- alert on dout = 15000 -----
91 tim_15000 <= (not cnt_out(0)) and (not cnt_out(1)) and (not cnt_out(2)) and
92             (cnt_out(3))    and (cnt_out(4))    and (not cnt_out(5)) and
93             (not cnt_out(6)) and (cnt_out(7))    and (not cnt_out(8)) and
94             (cnt_out(9))    and (not cnt_out(10)) and (cnt_out(11))    and
95             (cnt_out(12))    and (cnt_out(13));
96 -----
97
98 end rtl;

```

```

1  -- BRIEF DESCRIPTION: full adder (combination of 2 half adders)
2
3  library    ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity fulladder is
8  port
9  (
10     a      : in  std_logic;
11     b      : in  std_logic;
12     cin    : in  std_logic;
13     sum    : out std_logic;
14     cout   : out std_logic
15 );
16 end fulladder;
17
18 architecture behavior of fulladder is
19
20     -- COMPONENT: half adder -----
21     component halfadder is
22     port
23     (
24         a : in  std_logic;
25         b : in  std_logic;
26         s : out std_logic;
27         c : out std_logic
28     );
29     end component; -----
30
31     -- signals -----
32     signal ha1_sum  : std_logic;
33     signal ha1_cout : std_logic;
34     signal ha2_cout : std_logic;
35     -----
36
37     begin
38
39         ha1_inst: halfadder
40         port map
41         (
42             a => a,
43             b => b,
44             s => ha1_sum,
45             c => ha1_cout
46         );

```

```

47
48     ha2_inst: halfadder
49     port map
50     (
51         a => ha1_sum,
52         b => cin,
53         s => sum,
54         c => ha2_cout
55     );
56
57     cout <= ha1_cout or ha2_cout;
58
59 end behavior;

```

```

1  -- BRIEF DESCRIPTION: half adder
2
3  library    ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity halfadder is
8  port
9  (
10     a : in  std_logic;
11     b : in  std_logic;
12     s : out std_logic;
13     c : out std_logic
14 );
15 end halfadder;
16
17 architecture behavior of halfadder is
18
19     begin
20
21         s <= a XOR b;
22         c <= a AND b;
23
24 end behavior;

```

```

1  -- BRIEF DESCRIPTION: voter for DDR to SDR converter
2  -- COMMENTS:
3  -- it implements a majority decision among 8 different 1-bit inputs
4  -- it evaluates the sum of all the input bits and verifies if it is greater than 4
5  -- several pipeline layers are employed in order to make it possible to work with a 400 MHz clock
6
7  library    ieee;
8  use        ieee.std_logic_1164.all;
9  use        ieee.numeric_std.all;
10
11 entity voter is
12 port
13 (
14     clk      : in  std_logic;
15     clear_n  : in  std_logic;
16     enable   : in  std_logic;
17     d1       : in  std_logic;
18     d2       : in  std_logic;
19     d3       : in  std_logic;
20     d4       : in  std_logic;
21     d5       : in  std_logic;
22     d6       : in  std_logic;
23     d7       : in  std_logic;
24     d8       : in  std_logic;
25     gt4      : out std_logic;
26     eq4      : out std_logic
27 );
28 end voter;
29
30 architecture behavior of voter is
31

```



```

32 -- COMPONENT: half adder -----
33 component halfadder is
34 port
35 (
36     a : in  std_logic;
37     b : in  std_logic;
38     s : out std_logic;
39     c : out std_logic
40 );
41 end component; -----
42
43 -- COMPONENT: full adder -----
44 component fulladder is
45 port
46 (
47     a      : in  std_logic;
48     b      : in  std_logic;
49     cin    : in  std_logic;
50     sum    : out std_logic;
51     cout   : out std_logic
52 );
53 end component; -----
54
55 -- COMPONENT: register -----
56 component reg is
57 generic
58 (
59     N : integer := 8
60 );
61 port
62 (
63     clk       : in  std_logic;
64     enable    : in  std_logic;
65     clear_n   : in  std_logic; -- synchronous clear, active low
66     reset_n   : in  std_logic; -- asynchronous clear, active low
67     din       : in  std_logic_vector(N-1 downto 0);
68     dout      : out std_logic_vector(N-1 downto 0) := (others => '0')
69 );
70 end component; -----
71
72 -- SIGNALS -----
73 signal fa00_c      : std_logic;
74 signal fa00_s      : std_logic;
75 signal fa01_c      : std_logic;
76 signal fa01_s      : std_logic;
77 signal ha02_c      : std_logic;
78 signal ha02_s      : std_logic;
79 signal faw0_c      : std_logic;
80 signal faw0_s      : std_logic;
81 signal faw1_c      : std_logic;
82 signal faw1_s      : std_logic;
83 signal haw1_c      : std_logic;
84 signal haw1_s      : std_logic;
85 signal haw2_c      : std_logic;
86 signal haw2_s      : std_logic;
87 signal pipereg1_in : std_logic_vector(7 downto 0);
88 signal pipereg1_out : std_logic_vector(7 downto 0);
89 signal pipereg2_in : std_logic_vector(5 downto 0);
90 signal pipereg2_out : std_logic_vector(5 downto 0);
91 signal pipereg3_in : std_logic_vector(3 downto 0);
92 signal pipereg3_out : std_logic_vector(3 downto 0);
93 signal pipereg4_in : std_logic_vector(3 downto 0);
94 signal pipereg4_out : std_logic_vector(3 downto 0);
95 signal pipereg5_in : std_logic_vector(1 downto 0);
96 signal pipereg5_out : std_logic_vector(1 downto 0);
97 signal gt4_prepipe : std_logic;
98 signal eq4_prepipe : std_logic;
99 -----
100
101 begin
102
103     -- first pipeline layer -----
104     pipereg1_in(7) <= d8;
105     pipereg1_in(6) <= d7;
106     pipereg1_in(5) <= d6;
107     pipereg1_in(4) <= d5;

```

```

108 pipereg1_in(3) <= d4;
109 pipereg1_in(2) <= d3;
110 pipereg1_in(1) <= d2;
111 pipereg1_in(0) <= d1;
112 pipereg1: reg
113 generic map
114 (
115     N => 8
116 )
117 port map
118 (
119     clk      => clk,
120     enable   => enable,
121     clear_n  => clear_n,
122     reset_n  => '1',
123     din      => pipereg1_in,
124     dout     => pipereg1_out
125 ); -----
126
127 fa00: fulladder
128 port map
129 (
130     a      => pipereg1_out(0),
131     b      => pipereg1_out(1),
132     cin    => pipereg1_out(2),
133     sum    => fa00_s,
134     cout   => fa00_c
135 );
136
137 fa01: fulladder
138 port map
139 (
140     a      => pipereg1_out(3),
141     b      => pipereg1_out(4),
142     cin    => pipereg1_out(5),
143     sum    => fa01_s,
144     cout   => fa01_c
145 );
146
147 ha02: halfadder
148 port map
149 (
150     a => pipereg1_out(6),
151     b => pipereg1_out(7),
152     s => ha02_s,
153     c => ha02_c
154 );
155
156 -- second pipeline layer -----
157 pipereg2_in(5) <= ha02_s;
158 pipereg2_in(4) <= ha02_c;
159 pipereg2_in(3) <= fa01_s;
160 pipereg2_in(2) <= fa01_c;
161 pipereg2_in(1) <= fa00_s;
162 pipereg2_in(0) <= fa00_c;
163 pipereg2: reg
164 generic map
165 (
166     N => 6
167 )
168 port map
169 (
170     clk      => clk,
171     enable   => enable,
172     clear_n  => clear_n,
173     reset_n  => '1',
174     din      => pipereg2_in,
175     dout     => pipereg2_out
176 ); -----
177
178 faw0: fulladder
179 port map
180 (
181     a      => pipereg2_out(1),
182     b      => pipereg2_out(3),
183     cin    => pipereg2_out(5),

```

```

184     sum  => faw0_s,
185     cout => faw0_c
186 );
187
188 faw1: fulladder
189 port map
190 (
191     a    => pipereg2_out(0),
192     b    => pipereg2_out(2),
193     cin  => pipereg2_out(4),
194     sum  => faw1_s,
195     cout => faw1_c
196 );
197
198 -- third pipeline layer -----
199 pipereg3_in(3) <= faw1_c;
200 pipereg3_in(2) <= faw1_s;
201 pipereg3_in(1) <= faw0_c;
202 pipereg3_in(0) <= faw0_s;
203 pipereg3: reg
204 generic map
205 (
206     N => 4
207 )
208 port map
209 (
210     clk      => clk,
211     enable   => enable,
212     clear_n  => clear_n,
213     reset_n  => '1',
214     din      => pipereg3_in,
215     dout     => pipereg3_out
216 ); -----
217
218 haw1: halfadder
219 port map
220 (
221     a => pipereg3_out(1),
222     b => pipereg3_out(2),
223     s => haw1_s,
224     c => haw1_c
225 );
226
227 haw2: halfadder
228 port map
229 (
230     a => haw1_c,
231     b => pipereg3_out(3),
232     s => haw2_s,
233     c => haw2_c
234 );
235
236 -- fourth pipeline layer -----
237 pipereg4_in(3) <= haw2_c;
238 pipereg4_in(2) <= haw2_s;
239 pipereg4_in(1) <= haw1_s;
240 pipereg4_in(0) <= pipereg3_in(0);
241 pipereg4: reg
242 generic map
243 (
244     N => 4
245 )
246 port map
247 (
248     clk      => clk,
249     enable   => enable,
250     clear_n  => clear_n,
251     reset_n  => '1',
252     din      => pipereg4_in,
253     dout     => pipereg4_out
254 ); -----
255
256 gt4_prepipe <= ((pipereg4_out(0) or pipereg4_in(1)) and pipereg4_in(2)) or pipereg4_in(3);
257 eq4_prepipe <= (not pipereg4_out(0)) and (not pipereg4_in(1)) and pipereg4_in(2) and (not
258 pipereg4_in(3));

```

```

259  -- fifth pipeline layer -----
260  pipereg5_in(1) <= gt4_prepipes;
261  pipereg5_in(0) <= eq4_prepipes;
262  pipereg5: reg
263  generic map
264  (
265    N => 2
266  )
267  port map
268  (
269    clk      => clk,
270    enable   => enable,
271    clear_n  => clear_n,
272    reset_n  => '1',
273    din      => pipereg5_in,
274    dout     => pipereg5_out
275  ); -----
276
277  gt4 <= pipereg5_out(1);
278  eq4 <= pipereg5_out(0);
279
280  end behavior;

```

```

1  -- BRIEF DESCRIPTION: 8-bit DDR (msb-first) to 16-bit SDR converter
2  -- COMMENTS:
3  -- rws_in and DDR_in are DDR with respect to a 50 MHz clock
4  -- the converter works with a 400 MHz clock
5  -- rws_in and DDR_in are sampled on both positive and negative edges of the 400 MHz clock
6  -- 8 samples per clock level are collected with respect to the 50 MHz clock
7  -- 5 samples out of 8 are always correct
8  -- the detection of a variation of rws_in is based on a majority voting
9  -- the converter detects the variation of rws_in
10 -- the converter provides at its output an SDR version of the input data
11 -- the converter provides also a version of rws shifted of 2.5 ns with respect to the SDR data
12 -- deactivate the enable signal causes the loss of all the previously acquired samples
13
14 library ieee;
15 use ieee.std_logic_1164.all;
16 use ieee.numeric_std.all;
17
18 entity DDR_to_SDR_converter_EU is
19 port
20 (
21  -- clock and clear
22  clk_x8      : in  std_logic;
23  clear_n     : in  std_logic;
24  -- IO signals
25  rws_in      : in  std_logic;
26  rws_out     : out std_logic;
27  DDR_in      : in  std_logic_vector(7 downto 0);
28  SDR_out     : out std_logic_vector(15 downto 0);
29  -- control signals
30  system_clear_n : in  std_logic;
31  rwsngen_toggle : in  std_logic;
32  msb_enable    : in  std_logic;
33  lsb_enable    : in  std_logic;
34  -- status signals
35  clr_n        : out std_logic;
36  transition   : out std_logic
37 );
38 end DDR_to_SDR_converter_EU;
39
40 architecture rtl of DDR_to_SDR_converter_EU is
41
42  -- COMPONENT: positive-edge triggered type-D flip flop -----
43  component d_flipflop is
44  port
45  (
46    clk      : in  std_logic;
47    enable   : in  std_logic;
48    clear_n  : in  std_logic; -- synchronous clear, active low
49    reset_n  : in  std_logic; -- asynchronous reset, active low
50    din      : in  std_logic;
51    dout     : out std_logic

```

```

52 );
53 end component; -----
54
55 -- COMPONENT: negative-edge triggered type-D flip flop -----
56 component dff_negedge is
57 port
58 (
59   clk       : in  std_logic;
60   enable    : in  std_logic;
61   clear_n   : in  std_logic; -- synchronous clear, active low
62   reset_n   : in  std_logic; -- asynchronous reset, active low
63   din       : in  std_logic;
64   dout      : out std_logic
65 );
66 end component; -----
67
68 -- COMPONENT: positive-edge triggered register -----
69 component reg is
70 generic
71 (
72   N : integer := 8
73 );
74 port
75 (
76   clk       : in  std_logic;
77   enable    : in  std_logic;
78   clear_n   : in  std_logic; -- synchronous clear, active low
79   reset_n   : in  std_logic; -- asynchronous clear, active low
80   din       : in  std_logic_vector(N-1 downto 0);
81   dout      : out std_logic_vector(N-1 downto 0) := (others => '0')
82 );
83 end component; -----
84
85 -- COMPONENT: flipflop type-T (toggle) -----
86 component t_flipflop is
87 port
88 (
89   clk       : in  std_logic;
90   enable    : in  std_logic;
91   clear_n   : in  std_logic; -- synchronous clear, active low
92   reset_n   : in  std_logic; -- asynchronous reset, active low
93   din       : in  std_logic;
94   dout      : out std_logic := '0'
95 );
96 end component; -----
97
98 -- COMPONENT: voter -----
99 component voter is
100 port
101 (
102   clk       : in  std_logic;
103   clear_n   : in  std_logic;
104   enable    : in  std_logic;
105   d1        : in  std_logic;
106   d2        : in  std_logic;
107   d3        : in  std_logic;
108   d4        : in  std_logic;
109   d5        : in  std_logic;
110   d6        : in  std_logic;
111   d7        : in  std_logic;
112   d8        : in  std_logic;
113   gt4       : out std_logic;
114   eq4       : out std_logic
115 );
116 end component; -----
117
118 -- SIGNALS -----
119 signal pdff1_out      : std_logic;
120 signal pdff2_out      : std_logic;
121 signal pdff3_out      : std_logic;
122 signal pdff4_out      : std_logic;
123 signal ndff1_out      : std_logic;
124 signal ndff2_out      : std_logic;
125 signal ndff3_out      : std_logic;
126 signal ndff4_out      : std_logic;
127 signal pdff1_out_pipe : std_logic;

```

```

128 signal pdff2_out_pipe      : std_logic;
129 signal pdff3_out_pipe      : std_logic;
130 signal pdff4_out_pipe      : std_logic;
131 signal ndff1_out_pipe      : std_logic;
132 signal ndff2_out_pipe      : std_logic;
133 signal ndff3_out_pipe      : std_logic;
134 signal ndff4_out_pipe      : std_logic;
135 signal regp1_out           : std_logic_vector(7 downto 0);
136 signal regp2_out           : std_logic_vector(7 downto 0);
137 signal tracker_tgl        : std_logic;
138 signal tracker_out         : std_logic;
139 signal gt4                 : std_logic;
140 signal eq4                 : std_logic;
141 signal msb_out             : std_logic_vector(7 downto 0);
142 signal lsb_out             : std_logic_vector(7 downto 0);
143 signal rwdsgen_out         : std_logic;
144 signal pipereg1_out        : std_logic_vector(7 downto 0);
145 signal pipereg2_out        : std_logic_vector(7 downto 0);
146 signal pipereg3_out        : std_logic_vector(7 downto 0);
147 signal pipereg4_out        : std_logic_vector(7 downto 0);
148 signal pipereg5_out        : std_logic_vector(7 downto 0);
149 signal pipereg6_out        : std_logic_vector(7 downto 0);
150 signal pipereg7_out        : std_logic_vector(7 downto 0);
151 signal rwdsgen_toggle_pipe : std_logic;
152 -----
153
154 begin
155
156     clr_n <= clear_n;
157
158     -- posedge dff chain, 4th position -----
159     pdff4 : d_flipflop
160     port map
161     (
162         clk      => clk_x8,
163         enable   => '1',
164         clear_n  => system_clear_n,
165         reset_n  => '1',
166         din      => rwds_in,
167         dout     => pdff4_out
168     );
169     pdff4_pipe : d_flipflop
170     port map
171     (
172         clk      => clk_x8,
173         enable   => '1',
174         clear_n  => system_clear_n,
175         reset_n  => '1',
176         din      => pdff4_out,
177         dout     => pdff4_out_pipe
178     ); -----
179
180     -- posedge dff chain, 3th position -----
181     pdff3 : d_flipflop
182     port map
183     (
184         clk      => clk_x8,
185         enable   => '1',
186         clear_n  => system_clear_n,
187         reset_n  => '1',
188         din      => pdff4_out,
189         dout     => pdff3_out
190     );
191     pdff3_pipe : d_flipflop
192     port map
193     (
194         clk      => clk_x8,
195         enable   => '1',
196         clear_n  => system_clear_n,
197         reset_n  => '1',
198         din      => pdff3_out,
199         dout     => pdff3_out_pipe
200     ); -----
201
202     -- posedge dff chain, 2th position -----
203     pdff2 : d_flipflop

```

```

204 port map
205 (
206     clk         => clk_x8,
207     enable      => '1',
208     clear_n     => system_clear_n,
209     reset_n     => '1',
210     din         => pdff3_out,
211     dout        => pdff2_out
212 );
213 pdff2_pipe : d_flipflop
214 port map
215 (
216     clk         => clk_x8,
217     enable      => '1',
218     clear_n     => system_clear_n,
219     reset_n     => '1',
220     din         => pdff2_out,
221     dout        => pdff2_out_pipe
222 ); -----
223
224 -- posedge dff chain, 1th position -----
225 pdff1 : d_flipflop
226 port map
227 (
228     clk         => clk_x8,
229     enable      => '1',
230     clear_n     => system_clear_n,
231     reset_n     => '1',
232     din         => pdff2_out,
233     dout        => pdff1_out
234 );
235 pdff1_pipe : d_flipflop
236 port map
237 (
238     clk         => clk_x8,
239     enable      => '1',
240     clear_n     => system_clear_n,
241     reset_n     => '1',
242     din         => pdff1_out,
243     dout        => pdff1_out_pipe
244 ); -----
245
246 -- negedge dff chain, 4th position -----
247 ndff4 : dff_negedge
248 port map
249 (
250     clk         => clk_x8,
251     enable      => '1',
252     clear_n     => system_clear_n,
253     reset_n     => '1',
254     din         => rwds_in,
255     dout        => ndff4_out
256 );
257 ndff4_pipe : dff_negedge
258 port map
259 (
260     clk         => clk_x8,
261     enable      => '1',
262     clear_n     => system_clear_n,
263     reset_n     => '1',
264     din         => ndff4_out,
265     dout        => ndff4_out_pipe
266 ); -----
267
268 -- negedge dff chain, 3th position -----
269 ndff3 : dff_negedge
270 port map
271 (
272     clk         => clk_x8,
273     enable      => '1',
274     clear_n     => system_clear_n,
275     reset_n     => '1',
276     din         => ndff4_out,
277     dout        => ndff3_out
278 );
279 ndff3_pipe : dff_negedge

```

```

280 port map
281 (
282     clk      => clk_x8,
283     enable   => '1',
284     clear_n  => system_clear_n,
285     reset_n  => '1',
286     din      => ndff3_out,
287     dout     => ndff3_out_pipe
288 ); -----
289
290 -- negedge dff chain, 2th position -----
291 ndff2 : dff_negedge
292 port map
293 (
294     clk      => clk_x8,
295     enable   => '1',
296     clear_n  => system_clear_n,
297     reset_n  => '1',
298     din      => ndff3_out,
299     dout     => ndff2_out
300 );
301 ndff2_pipe : dff_negedge
302 port map
303 (
304     clk      => clk_x8,
305     enable   => '1',
306     clear_n  => system_clear_n,
307     reset_n  => '1',
308     din      => ndff2_out,
309     dout     => ndff2_out_pipe
310 ); -----
311
312 -- negedge dff chain, 1th position -----
313 ndff1 : dff_negedge
314 port map
315 (
316     clk      => clk_x8,
317     enable   => '1',
318     clear_n  => system_clear_n,
319     reset_n  => '1',
320     din      => ndff2_out,
321     dout     => ndff1_out
322 );
323 ndff1_pipe : dff_negedge
324 port map
325 (
326     clk      => clk_x8,
327     enable   => '1',
328     clear_n  => system_clear_n,
329     reset_n  => '1',
330     din      => ndff1_out,
331     dout     => ndff1_out_pipe
332 ); -----
333
334 -- voter -----
335 voter_inst: voter
336 port map
337 (
338     clk      => clk_x8,
339     clear_n  => system_clear_n,
340     enable   => '1',
341     d1       => ndff4_out_pipe,
342     d2       => pdff4_out_pipe,
343     d3       => ndff3_out_pipe,
344     d4       => pdff3_out_pipe,
345     d5       => ndff2_out_pipe,
346     d6       => pdff2_out_pipe,
347     d7       => ndff1_out_pipe,
348     d8       => pdff1_out_pipe,
349     gt4      => gt4,
350     eq4      => eq4
351 ); -----
352
353 tracker_tgl <= (gt4 xor tracker_out) and (not eq4);
354
355 -- result tracker -----

```



```

356 tracker: t_flipflop
357 port map
358 (
359     clk      => clk_x8,
360     enable   => '1',
361     clear_n  => system_clear_n,
362     reset_n  => '1',
363     din      => tracker_tgl,
364     dout     => tracker_out
365 ); -----
366
367 -- data register chain - register 1 -----
368 regp1: reg
369 generic map
370 (
371     N => 8
372 )
373 port map
374 (
375     clk      => clk_x8,
376     enable   => '1',
377     clear_n  => system_clear_n,
378     reset_n  => '1',
379     din      => DDR_in,
380     dout     => regp1_out
381 ); -----
382
383 -- data register chain - register 2 -----
384 regp2: reg
385 generic map
386 (
387     N => 8
388 )
389 port map
390 (
391     clk      => clk_x8,
392     enable   => '1',
393     clear_n  => system_clear_n,
394     reset_n  => '1',
395     din      => regp1_out,
396     dout     => regp2_out
397 ); -----
398
399 -- data register chain - pipeline register 1 -----
400 pipereg1: reg
401 generic map
402 (
403     N => 8
404 )
405 port map
406 (
407     clk      => clk_x8,
408     enable   => '1',
409     clear_n  => system_clear_n,
410     reset_n  => '1',
411     din      => regp2_out,
412     dout     => pipereg1_out
413 ); -----
414
415 -- data register chain - pipeline register 2 -----
416 pipereg2: reg
417 generic map
418 (
419     N => 8
420 )
421 port map
422 (
423     clk      => clk_x8,
424     enable   => '1',
425     clear_n  => system_clear_n,
426     reset_n  => '1',
427     din      => pipereg1_out,
428     dout     => pipereg2_out
429 ); -----
430
431 -- data register chain - pipeline register 3 -----

```

```

432 pipereg3: reg
433 generic map
434 (
435     N => 8
436 )
437 port map
438 (
439     clk      => clk_x8,
440     enable   => '1',
441     clear_n  => system_clear_n,
442     reset_n  => '1',
443     din      => pipereg2_out,
444     dout     => pipereg3_out
445 ); -----
446
447 -- data register chain - pipeline register 4 -----
448 pipereg4: reg
449 generic map
450 (
451     N => 8
452 )
453 port map
454 (
455     clk      => clk_x8,
456     enable   => '1',
457     clear_n  => system_clear_n,
458     reset_n  => '1',
459     din      => pipereg3_out,
460     dout     => pipereg4_out
461 ); -----
462
463 -- data register chain - pipeline register 5 -----
464 pipereg5: reg
465 generic map
466 (
467     N => 8
468 )
469 port map
470 (
471     clk      => clk_x8,
472     enable   => '1',
473     clear_n  => system_clear_n,
474     reset_n  => '1',
475     din      => pipereg4_out,
476     dout     => pipereg5_out
477 ); -----
478
479 -- data register chain - pipeline register 6 -----
480 pipereg6: reg
481 generic map
482 (
483     N => 8
484 )
485 port map
486 (
487     clk      => clk_x8,
488     enable   => '1',
489     clear_n  => system_clear_n,
490     reset_n  => '1',
491     din      => pipereg5_out,
492     dout     => pipereg6_out
493 ); -----
494
495 -- data register chain - pipeline register 7 -----
496 pipereg7: reg
497 generic map
498 (
499     N => 8
500 )
501 port map
502 (
503     clk      => clk_x8,
504     enable   => '1',
505     clear_n  => system_clear_n,
506     reset_n  => '1',
507     din      => pipereg6_out,

```

```

508     dout      => pipereg7_out
509 ); -----
510
511 -- msb register -----
512 msb: reg
513 generic map
514 (
515     N => 8
516 )
517 port map
518 (
519     clk      => clk_x8,
520     enable   => msb_enable,
521     clear_n  => system_clear_n,
522     reset_n  => '1',
523     din      => pipereg7_out,
524     dout     => msb_out
525 ); -----
526
527 -- lsb register -----
528 lsb: reg
529 generic map
530 (
531     N => 8
532 )
533 port map
534 (
535     clk      => clk_x8,
536     enable   => lsb_enable,
537     clear_n  => system_clear_n,
538     reset_n  => '1',
539     din      => pipereg7_out,
540     dout     => lsb_out
541 ); -----
542
543 SDR_out(15 downto 8) <= msb_out;
544 SDR_out(7  downto 0) <= lsb_out;
545 transition <= tracker_tgl;
546
547 -- rwdsgen generation pipelining -----
548 rwdsgen_pipe: d_flipflop
549 port map
550 (
551     clk      => clk_x8,
552     enable   => '1',
553     clear_n  => system_clear_n,
554     reset_n  => '1',
555     din      => rwdsgen_toggle,
556     dout     => rwdsgen_toggle_pipe
557 ); -----
558
559 -- shifted-rwds generator -----
560 rwdsgen: t_flipflop
561 port map
562 (
563     clk      => clk_x8,
564     enable   => '1',
565     clear_n  => system_clear_n,
566     reset_n  => '1',
567     din      => rwdsgen_toggle_pipe,
568     dout     => rwdsgen_out
569 ); -----
570
571 rwds_out <= not rwdsgen_out;
572
573 end rtl;

```

```

1 -- BRIEF DESCRIPTION: DDR_to_SDR_converter control unit
2 -- COMMENTS:
3 -- use it together with DDR_to_SDR_converter_EU.vhd to create DDR_to_SDR_converter.vhd
4 -- the output (control signals) generation is registered to improve the performace
5
6 library ieee;
7 use      ieee.std_logic_1164.all;

```

```

8 use      ieee.numeric_std.all;
9
10 entity DDR_to_SDR_converter_CU is
11 port
12 (
13     -- clock, reset and clear
14     clk_x8      : in  std_logic;
15     clr_n       : in  std_logic;
16     -- status signals
17     transition  : in  std_logic;
18     -- control signals
19     system_clear_n : out std_logic;
20     rwdsgen_toggle : out std_logic;
21     msb_enable   : out std_logic;
22     lsb_enable   : out std_logic
23 );
24 end entity DDR_to_SDR_converter_CU;
25
26 architecture fsm of DDR_to_SDR_converter_CU is
27
28     -- states definition -----
29     type state is
30     (
31         reset,
32         reset_wait,
33         idle,
34         idle_intra,
35         msb_tx,
36         lsb_tx
37     ); -----
38
39     -- states declaration -----
40     signal present_state : state;
41     signal next_state    : state;
42     -----
43
44     -- unregistered control signals -----
45     signal int_system_clear_n : std_logic;
46     signal int_rwdsgen_toggle : std_logic;
47     signal int_msb_enable     : std_logic;
48     signal int_lsb_enable     : std_logic;
49     -----
50
51 begin
52
53     -- evaluation of the next state -----
54     next_state_evaluation: process
55     (
56         -- sensitivity list
57         present_state,
58         transition
59     )
60     begin
61         case present_state is
62             -----
63             when reset =>
64                 next_state <= reset_wait;
65             -----
66             when reset_wait | idle | lsb_tx =>
67                 if (transition = '1') then
68                     next_state <= msb_tx;
69                 else
70                     next_state <= idle;
71                 end if;
72             -----
73             when msb_tx =>
74                 next_state <= idle_intra;
75             -----
76             when idle_intra =>
77                 if (transition = '1') then
78                     next_state <= lsb_tx;
79                 else
80                     next_state <= idle_intra;
81                 end if;
82             -----
83             when others =>

```

```

84     next_state <= reset;
85     -----
86     end case;
87 end process next_state_evaluation; -----
88
89 -- state transition -----
90 state_transition: process (clk_x8, clr_n)
91 begin
92     if (rising_edge(clk_x8)) then
93         if (clr_n = '0') then
94             present_state <= reset;
95         else
96             present_state <= next_state;
97         end if;
98     end if;
99 end process state_transition; -----
100
101 -- control signals definition -----
102 control_signals_definition: process (present_state)
103 begin
104     -- default values -----
105     int_system_clear_n    <= '1';
106     int_rwdsgen_toggle    <= '0';
107     int_msb_enable        <= '0';
108     int_lsb_enable        <= '0';
109     -----
110     case present_state is
111     -----
112     when reset =>
113         int_system_clear_n <= '0';
114     -----
115     when reset_wait =>
116     -----
117     when idle | idle_intra =>
118     -----
119     when msb_tx =>
120         int_msb_enable      <= '1';
121         int_rwdsgen_toggle <= '1';
122     -----
123     when lsb_tx =>
124         int_lsb_enable      <= '1';
125         int_rwdsgen_toggle <= '1';
126     -----
127     end case;
128 end process control_signals_definition; -----
129
130 -- control signals pipelining -----
131 control_pipe: process (clk_x8)
132 begin
133     if (rising_edge(clk_x8)) then
134         system_clear_n    <= int_system_clear_n;
135         rwdsgen_toggle    <= int_rwdsgen_toggle;
136         msb_enable        <= int_msb_enable;
137         lsb_enable        <= int_lsb_enable;
138     end if;
139 end process control_pipe; -----
140
141 end architecture fsm;

```

```

1  -- BRIEF DESCRIPTION: 8-bit DDR (msb-first) to 16-bit SDR converter
2  -- COMMENTS:
3  -- rwd_in and DDR_in are DDR with respect to a 50 MHz clock
4  -- the converter works with a 400 MHz clock
5  -- rwd_in and DDR_in are sampled on both positive and negative edges of the 400 MHz clock
6  -- 8 samples per clock level are collected with respect to the 50 MHz clock
7  -- 5 samples out of 8 are always correct
8  -- the detection of a variation of rwd_in is based on a majority voting
9  -- the converter detects the variation of rwd_in
10 -- the converter provides at its output an SDR version of the input data
11 -- the converter provides also a version of rwd shifted of 2.5 ns with respect to the SDR data
12 -- deactivate the enable signal causes the loss of all the previously acquired samples
13
14 library    ieee;
15 use        ieee.std_logic_1164.all;

```

```

16 use      ieee.numeric_std.all;
17
18 entity DDR_to_SDR_converter is
19 port
20 (
21   -- clock, reset and clear
22   clk_x8      : in  std_logic;
23   clear_n     : in  std_logic;
24   -- IO signals
25   rwds_in     : in  std_logic;
26   rwds_out    : out std_logic;
27   DDR_in      : in  std_logic_vector(7 downto 0);
28   SDR_out     : out std_logic_vector(15 downto 0)
29 );
30 end DDR_to_SDR_converter;
31
32 architecture rtl of DDR_to_SDR_converter is
33
34   -- COMPONENT: execution unit -----
35   component DDR_to_SDR_converter_EU is
36   port
37   (
38     -- clock, reset and clear
39     clk_x8      : in  std_logic;
40     clear_n     : in  std_logic;
41     -- IO signals
42     rwds_in     : in  std_logic;
43     rwds_out    : out std_logic;
44     DDR_in      : in  std_logic_vector(7 downto 0);
45     SDR_out     : out std_logic_vector(15 downto 0);
46     -- control signals
47     system_clear_n : in  std_logic;
48     rwdsgen_toggle : in  std_logic;
49     msb_enable    : in  std_logic;
50     lsb_enable    : in  std_logic;
51     -- status signals
52     clr_n        : out std_logic;
53     transition   : out std_logic
54   );
55   end component; -----
56
57   -- COMPONENT: control unit -----
58   component DDR_to_SDR_converter_CU is
59   port
60   (
61     -- clock, reset and clear
62     clk_x8      : in  std_logic;
63     clr_n       : in  std_logic;
64     -- status signals
65     transition  : in  std_logic;
66     -- control signals
67     system_clear_n : out std_logic;
68     rwdsgen_toggle : out std_logic;
69     msb_enable    : out std_logic;
70     lsb_enable    : out std_logic
71   );
72   end component; -----
73
74   -- SIGNALS -----
75   signal system_clear_n : std_logic;
76   signal rwdsgen_toggle : std_logic;
77   signal msb_enable     : std_logic;
78   signal lsb_enable     : std_logic;
79   signal clr_n          : std_logic;
80   signal transition     : std_logic;
81   -----
82
83   begin
84
85     -- EU instance -----
86     EU: DDR_to_SDR_converter_EU
87     port map
88     (
89       clk_x8      => clk_x8,
90       clear_n     => clear_n,
91       rwds_in     => rwds_in,

```

```

92     rwds_out          => rwds_out ,
93     DDR_in           => DDR_in ,
94     SDR_out          => SDR_out ,
95     system_clear_n   => system_clear_n ,
96     rwdsgen_toggle   => rwdsgen_toggle ,
97     msb_enable       => msb_enable ,
98     lsb_enable       => lsb_enable ,
99     clr_n            => clr_n ,
100    transition        => transition
101 ); -----
102
103 -- CU instance -----
104 CU: DDR_to_SDR_converter_CU
105 port map
106 (
107     clk_x8           => clk_x8 ,
108     clr_n            => clr_n ,
109     transition       => transition ,
110     system_clear_n   => system_clear_n ,
111     rwdsgen_toggle   => rwdsgen_toggle ,
112     msb_enable       => msb_enable ,
113     lsb_enable       => lsb_enable
114 ); -----
115
116 end rtl;

```

```

1 -- BRIEF DESCRIPTION: decoder with a 2-bit input (i.e. 4 decoded outputs)
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5 use ieee.numeric_std.all;
6
7 entity decoder_2bit is
8 port
9 (
10    code    : in    std_logic_vector(1 downto 0);
11    dec00   : out   std_logic;
12    dec01   : out   std_logic;
13    dec10   : out   std_logic;
14    dec11   : out   std_logic
15 );
16 end decoder_2bit;
17
18 architecture behavior of decoder_2bit is
19
20     begin
21
22         -- main process -----
23         output_evaluation: process (code)
24         begin
25             case code is
26                 when "00" =>
27                 dec00 <= '1';
28                 dec01 <= '0';
29                 dec10 <= '0';
30                 dec11 <= '0';
31                 when "01" =>
32                 dec00 <= '0';
33                 dec01 <= '1';
34                 dec10 <= '0';
35                 dec11 <= '0';
36                 when "10" =>
37                 dec00 <= '0';
38                 dec01 <= '0';
39                 dec10 <= '1';
40                 dec11 <= '0';
41                 when others =>
42                 dec00 <= '0';
43                 dec01 <= '0';
44                 dec10 <= '0';
45                 dec11 <= '1';
46             end case;
47         end process output_evaluation; -----
48

```

```

49 end behavior;

1  -- BRIEF DESCRIPTION: synchronizer execution unit
2  -- COMMENTS:
3  -- use it together with synchronizer_CU.vhd to create synchronizer.vhd
4
5  library    ieee;
6  use        ieee.std_logic_1164.all;
7  use        ieee.numeric_std.all;
8
9  entity synchronizer_EU is
10 port
11 (
12     -- clock and reset
13     clk           : in    std_logic;
14     rst_n         : in    std_logic;
15     -- data signals
16     synch_enable  : in    std_logic;
17     synch_clear_n : in    std_logic;
18     synch_strobe  : in    std_logic;
19     synch_validout : out   std_logic;
20     synch_busy    : out   std_logic;
21     synch_din     : in    std_logic_vector(15 downto 0);
22     synch_dout    : out   std_logic_vector(15 downto 0);
23     burstcount    : in    std_logic_vector(10 downto 0);
24     counter_enable : in    std_logic;
25     counter_clear_n : in   std_logic;
26     counter_up_downN : in   std_logic;
27     counter_out   : out   std_logic_vector(10 downto 0);
28     -- control signals
29     system_clear_n : in    std_logic;
30     system_enable  : in    std_logic;
31     burstlen_enable : in    std_logic;
32     burstlen_counter_enable : in  std_logic;
33     outreg_enable  : in    std_logic;
34     data_counter_enable : in  std_logic;
35     busy           : in    std_logic;
36     validout      : in    std_logic;
37     outpipe_enable : in    std_logic;
38     outpipe_clear_n : in   std_logic;
39     -- status signals
40     burst_end     : out   std_logic;
41     start_sampling : out   std_logic;
42     enable        : out   std_logic;
43     clear_n       : out   std_logic
44 );
45 end synchronizer_EU;
46
47 architecture rtl of synchronizer_EU is
48
49     -- COMPONENT: N-bit register -----
50     component reg is
51     generic
52     (
53         N : integer := 8
54     );
55     port
56     (
57         clk       : in  std_logic;
58         enable    : in  std_logic;
59         clear_n   : in  std_logic; -- synchronous clear, active low
60         reset_n  : in  std_logic; -- asynchronous clear, active low
61         din      : in  std_logic_vector(N-1 downto 0);
62         dout     : out std_logic_vector(N-1 downto 0) := (others => '0')
63     );
64     end component; -----
65
66     -- COMPONENT: set-reset flipflop -----
67     component sr_flipflop is
68     port
69     (
70         clk       : in  std_logic;
71         set       : in  std_logic;
72         clear_n   : in  std_logic;

```



```

73     rst_n      : in std_logic;
74     dout       : out std_logic
75 );
76 end component; -----
77
78 -- COMPONENT: flipflop type D -----
79 component d_flipflop is
80 port
81 (
82     clk        : in  std_logic;
83     enable     : in  std_logic;
84     clear_n    : in  std_logic; -- synchronous clear, active low
85     reset_n    : in  std_logic; -- asynchronous reset, active low
86     din        : in  std_logic;
87     dout       : out std_logic
88 );
89 end component; -----
90
91 -- COMPONENT: multiplexer 4-inputs 1-output N-bit -----
92 component mux_4to1 is
93 generic
94 (
95     N : integer := 1
96 );
97 port
98 (
99     din_00     : in    std_logic_vector((N-1) downto 0);
100    din_01     : in    std_logic_vector((N-1) downto 0);
101    din_10     : in    std_logic_vector((N-1) downto 0);
102    din_11     : in    std_logic_vector((N-1) downto 0);
103    sel        : in    std_logic_vector(1 downto 0);
104    dout       : out   std_logic_vector((N-1) downto 0)
105 );
106 end component; -----
107
108 -- COMPONENT: comparison between two inputs (N bit) -----
109 component comparator_Nbit is
110 generic
111 (
112     N : integer := 1
113 );
114 port
115 (
116     din_0      : in    std_logic_vector((N-1) downto 0);
117     din_1      : in    std_logic_vector((N-1) downto 0);
118     equal      : out   std_logic
119 );
120 end component; -----
121
122 -- COMPONENT: decoder with a 2-bit input (i.e. 4 decoded outputs) -----
123 component decoder_2bit is
124 port
125 (
126     code       : in    std_logic_vector(1 downto 0);
127     dec00      : out   std_logic;
128     dec01      : out   std_logic;
129     dec10      : out   std_logic;
130     dec11      : out   std_logic
131 );
132 end component; -----
133
134 -- COMPONENT: synchronous up-counter N bit -----
135 component counter_Nbit is
136 generic
137 (
138     N : integer := 4
139 );
140 port
141 (
142     clk        : in  std_logic;
143     enable     : in  std_logic;
144     clear_n    : in  std_logic;
145     reset_n    : in  std_logic;
146     dout       : out std_logic_vector(N-1 downto 0)
147 );
148 end component; -----

```

```

149 -- COMPONENT: synchronous up-down counter 11 bit -----
150 component counter_11bit_updown is
151 port
152 (
153     clock      : in  std_logic ;
154     cnt_en     : in  std_logic ;
155     sclr      : in  std_logic ;
156     updown    : in  std_logic ;
157     q         : out std_logic_vector (10 downto 0)
158 );
159 end component; -----
160
161
162 -- SIGNALS -----
163 signal code_counter_out      : std_logic_vector(1 downto 0);
164 signal dec00                 : std_logic;
165 signal dec01                 : std_logic;
166 signal dec10                 : std_logic;
167 signal dec11                 : std_logic;
168 signal din00_out             : std_logic_vector(15 downto 0);
169 signal din01_out             : std_logic_vector(15 downto 0);
170 signal din10_out             : std_logic_vector(15 downto 0);
171 signal din11_out             : std_logic_vector(15 downto 0);
172 signal din00_enable          : std_logic;
173 signal datamux_out           : std_logic_vector(15 downto 0);
174 signal burstlen_reg_out      : std_logic_vector(10 downto 0);
175 signal burstlen_counter_out  : std_logic_vector(10 downto 0);
176 signal effective_burstlen_cnt_en : std_logic;
177 signal effective_burstlen_cnt_clear : std_logic;
178 signal effective_burstlen_cnt_up_downN : std_logic;
179 signal data_counter_out      : std_logic_vector(1 downto 0);
180 signal outreg_out            : std_logic_vector(15 downto 0);
181 -----
182
183 begin
184
185     clear_n <= synch_clear_n;
186     enable <= synch_enable;
187     counter_out <= burstlen_counter_out;
188
189     din00_enable <= dec00 and system_enable;
190     effective_burstlen_cnt_en <= burstlen_counter_enable or counter_enable;
191     effective_burstlen_cnt_clear <= not( system_clear_n and counter_clear_n );
192     effective_burstlen_cnt_up_downN <= burstlen_counter_enable or counter_up_downN;
193
194     -- code counter -----
195     code_counter : counter_Nbit
196     generic map
197     (
198         N => 2
199     )
200     port map
201     (
202         clk      => synch_strobe,
203         enable   => system_enable,
204         clear_n  => system_clear_n,
205         reset_n  => rst_n,
206         dout     => code_counter_out
207     ); -----
208
209     -- synchronizer (synchronizing flipflop) -----
210     synchronizer : sr_flipflop
211     port map
212     (
213         clk      => synch_strobe,
214         set       => system_enable,
215         clear_n  => system_clear_n,
216         rst_n    => rst_n,
217         dout     => start_sampling
218     ); -----
219
220     -- enabler (enabling decoder) -----
221     dec : decoder_2bit
222     port map
223     (
224         code => code_counter_out,

```

```

225     dec00 => dec00,
226     dec01 => dec01,
227     dec10 => dec10,
228     dec11 => dec11
229 ); -----
230
231 -- din11 register -----
232 din11 : reg
233 generic map
234 (
235     N => 16
236 )
237 port map
238 (
239     clk      => synch_strobe,
240     enable   => dec11,
241     clear_n  => '1',
242     reset_n  => '1',
243     din      => synch_din,
244     dout     => din11_out
245 ); -----
246
247 -- din10 register -----
248 din10 : reg
249 generic map
250 (
251     N => 16
252 )
253 port map
254 (
255     clk      => synch_strobe,
256     enable   => dec10,
257     clear_n  => '1',
258     reset_n  => '1',
259     din      => synch_din,
260     dout     => din10_out
261 ); -----
262
263 -- din01 register -----
264 din01 : reg
265 generic map
266 (
267     N => 16
268 )
269 port map
270 (
271     clk      => synch_strobe,
272     enable   => dec01,
273     clear_n  => '1',
274     reset_n  => '1',
275     din      => synch_din,
276     dout     => din01_out
277 ); -----
278
279 -- din00 register -----
280 din00 : reg
281 generic map
282 (
283     N => 16
284 )
285 port map
286 (
287     clk      => synch_strobe,
288     enable   => din00_enable,
289     clear_n  => '1',
290     reset_n  => '1',
291     din      => synch_din,
292     dout     => din00_out
293 ); -----
294
295 -- data mux -----
296 datamux : mux_4to1
297 generic map
298 (
299     N => 16
300 )

```

```

301 port map
302 (
303     din_00 => din00_out,
304     din_01 => din01_out,
305     din_10 => din10_out,
306     din_11 => din11_out,
307     sel    => data_counter_out,
308     dout   => datamux_out
309 ); -----
310
311 -- outreg (output register) -----
312 outreg : reg
313 generic map
314 (
315     N => 16
316 )
317 port map
318 (
319     clk      => clk,
320     enable   => outreg_enable,
321     clear_n  => '1',
322     reset_n  => '1',
323     din      => datamux_out,
324     dout     => outreg_out
325 ); -----
326
327 -- burstlen register -----
328 burstlen : reg
329 generic map
330 (
331     N => 11
332 )
333 port map
334 (
335     clk      => synch_strobe,
336     enable   => burstlen_enable,
337     clear_n  => '1',
338     reset_n  => '1',
339     din      => burstcount,
340     dout     => burstlen_reg_out
341 ); -----
342
343 -- burstlen_counter -----
344 burstlen_counter : counter_11bit_updown
345 port map
346 (
347     clock    => clk,
348     cnt_en   => effective_burstlen_cnt_en,
349     sclr     => effective_burstlen_cnt_clear,
350     updown   => effective_burstlen_cnt_up_downN,
351     q        => burstlen_counter_out
352 ); -----
353
354 -- burstlen comparator -----
355 burstlen_cmp : comparator_Nbit
356 generic map
357 (
358     N => 11
359 )
360 port map
361 (
362     din_0 => burstlen_reg_out,
363     din_1 => burstlen_counter_out,
364     equal => burst_end
365 ); -----
366
367 -- data counter -----
368 data_counter : counter_Nbit
369 generic map
370 (
371     N => 2
372 )
373 port map
374 (
375     clk      => clk,
376     enable   => data_counter_enable,

```

```

377     clear_n    => system_clear_n,
378     reset_n    => '1',
379     dout       => data_counter_out
380 ); -----
381
382 -- validout pipeline -----
383 valid_pipe : d_flipflop
384 port map
385 (
386     clk        => clk,
387     enable     => outpipe_enable,
388     clear_n    => outpipe_clear_n,
389     reset_n    => '1',
390     din        => validout,
391     dout       => synch_validout
392 ); -----
393
394 -- busy pipeline -----
395 busy_pipe : d_flipflop
396 port map
397 (
398     clk        => clk,
399     enable     => outpipe_enable,
400     clear_n    => outpipe_clear_n,
401     reset_n    => '1',
402     din        => busy,
403     dout       => synch_busy
404 ); -----
405
406 -- outreg pipeline -----
407 outpipe : reg
408 generic map
409 (
410     N => 16
411 )
412 port map
413 (
414     clk        => clk,
415     enable     => outpipe_enable,
416     clear_n    => outpipe_clear_n,
417     reset_n    => '1',
418     din        => outreg_out,
419     dout       => synch_dout
420 ); -----
421
422 end rtl;

```

```

1  -- BRIEF DESCRIPTION: synchronizer control unit
2  -- COMMENTS:
3  -- use it together with synchronizer_EU.vhd to create synchronizer.vhd
4
5  library    ieee;
6  use        ieee.std_logic_1164.all;
7  use        ieee.numeric_std.all;
8
9  entity synchronizer_CU is
10 port
11 (
12     -- clock and reset
13     clk          : in    std_logic;
14     rst_n        : in    std_logic;
15     -- status signals
16     burst_end    : in    std_logic;
17     start_sampling : in    std_logic;
18     enable       : in    std_logic;
19     clear_n      : in    std_logic;
20     -- control signals
21     system_clear_n : out   std_logic;
22     system_enable  : out   std_logic;
23     burstlen_enable : out   std_logic;
24     burstlen_counter_enable : out std_logic;
25     outreg_enable  : out   std_logic;
26     data_counter_enable : out std_logic;
27     busy          : out   std_logic;

```

```

28   validout           : out   std_logic;
29   outpipe_enable     : out   std_logic;
30   outpipe_clear_n    : out   std_logic
31 );
32 end entity synchronizer_CU;
33
34 architecture fsm of synchronizer_CU is
35
36   -- states definition -----
37   type state is
38   (
39     reset,
40     idle,
41     idle_disabled,
42     reception_init,
43     reception,
44     idle_clear
45   ); -----
46
47   -- states declaration -----
48   signal present_state : state;
49   signal next_state    : state;
50   -----
51
52   begin
53
54     -- evaluation of the next state -----
55     next_state_evaluation: process
56     (
57       -- sensitivity list
58       present_state,
59       burst_end,
60       start_sampling,
61       enable
62     )
63     begin
64       case present_state is
65         -----
66         when reset =>
67           if (enable = '1') then
68             next_state <= idle;
69           else
70             next_state <= idle_disabled;
71           end if;
72         -----
73         when idle =>
74           if (start_sampling = '1') then
75             next_state <= reception_init;
76           else
77             next_state <= idle;
78           end if;
79         -----
80         when idle_disabled =>
81           if (enable = '1') then
82             next_state <= idle;
83           else
84             next_state <= idle_disabled;
85           end if;
86         -----
87         when reception_init =>
88           next_state <= reception;
89         -----
90         when reception =>
91           if (burst_end = '1') then
92             next_state <= idle_clear;
93           else
94             next_state <= reception;
95           end if;
96         -----
97         when idle_clear =>
98           next_state <= idle_clear;
99         -----
100        when others =>
101          next_state <= reset;
102        -----
103     end case;

```

```

104 end process next_state_evaluation; -----
105
106 -- state transition -----
107 state_transition: process (clk, rst_n, clear_n)
108 begin
109     if (rst_n = '0') then
110         present_state <= reset;
111     elsif (rising_edge(clk)) then
112         if (clear_n = '0') then
113             present_state <= reset;
114         else
115             present_state <= next_state;
116         end if;
117     end if;
118 end process state_transition; -----
119
120 -- control signals definition -----
121 control_signals_definition: process (present_state)
122 begin
123     -- default values -----
124     system_clear_n      <= '1';
125     system_enable       <= '0';
126     burstlen_enable     <= '0';
127     burstlen_counter_enable <= '0';
128     outreg_enable       <= '0';
129     data_counter_enable <= '0';
130     busy                <= '0';
131     validout            <= '0';
132     outpipe_enable      <= '0';
133     outpipe_clear_n    <= '1';
134     -----
135     case present_state is
136     -----
137     when reset =>
138         system_clear_n <= '0';
139         outpipe_clear_n <= '0';
140     -----
141     when idle =>
142         system_enable <= '1';
143         burstlen_enable <= '1';
144         outpipe_enable <= '1';
145     -----
146     when idle_disabled =>
147     -----
148     when reception_init =>
149         busy <= '1';
150         system_enable <= '1';
151         outpipe_enable <= '1';
152         outreg_enable <= '1';
153         data_counter_enable <= '1';
154         burstlen_counter_enable <= '1';
155     -----
156     when reception =>
157         busy <= '1';
158         system_enable <= '1';
159         outpipe_enable <= '1';
160         outreg_enable <= '1';
161         data_counter_enable <= '1';
162         burstlen_counter_enable <= '1';
163         validout <= '1';
164     -----
165     when idle_clear =>
166         system_clear_n <= '0';
167         outpipe_enable <= '1';
168     -----
169     end case;
170 end process control_signals_definition; -----
171
172 end architecture fsm;

```

```

1 -- BRIEF DESCRIPTION: synchronization of synch_din (synchronous with synch_strobe) with clk
2 -- COMMENTS:
3 -- the rising edge of synch_strobe is center aligned with synch_din
4 -- the delay between synch_strobe and clk is unknown

```

```

5  -- synch_din is synchronous with synch_strobe
6  -- synch_dout is exactly equal to synch_din but synchrononus with clk
7  -- burstcount represents the number of consecutive data to be synchronized
8  -- burstcount is sampled by synch_strobe
9  -- burstcount must be valid before synch_strobe starts oscillating
10 -- an operation is terminated when synch_busy goes low after going high
11 -- synch_busy goes low when burstcount is reached, every data provided after this event is ignored
12 -- at the end of an operation, the synchronizer must be cleared in order to start a new operation
13 -- synch_enable must be set to '1' before synch_strobe starts oscillating
14 -- it is recommended to keep synch_enable set to '1' up to the end of the synchronization
15 -- the internal 11-bit updown counter can be accessed from outside
16 -- it is recommended to access to the 11-bit updown counter only when synch_enable is low
17
18 library    ieee;
19 use        ieee.std_logic_1164.all;
20 use        ieee.numeric_std.all;
21
22 entity synchronizer is
23 port
24 (
25     clk                : in    std_logic;
26     rst_n              : in    std_logic;
27     synch_enable       : in    std_logic;
28     synch_clear_n      : in    std_logic;
29     synch_strobe       : in    std_logic;
30     synch_validout     : out   std_logic;
31     synch_busy         : out   std_logic;
32     synch_din          : in    std_logic_vector(15 downto 0);
33     synch_dout         : out   std_logic_vector(15 downto 0);
34     burstcount         : in    std_logic_vector(10 downto 0);
35     counter_enable     : in    std_logic;
36     counter_clear_n    : in    std_logic;
37     counter_up_downN   : in    std_logic;
38     counter_out        : out   std_logic_vector(10 downto 0)
39 );
40 end synchronizer;
41
42 architecture rtl of synchronizer is
43
44     -- COMPONENT: execution unit -----
45     component synchronizer_EU is
46     port
47     (
48         -- clock and reset
49         clk                : in    std_logic;
50         rst_n              : in    std_logic;
51         -- data signals
52         synch_enable       : in    std_logic;
53         synch_clear_n      : in    std_logic;
54         synch_strobe       : in    std_logic;
55         synch_validout     : out   std_logic;
56         synch_busy         : out   std_logic;
57         synch_din          : in    std_logic_vector(15 downto 0);
58         synch_dout         : out   std_logic_vector(15 downto 0);
59         burstcount         : in    std_logic_vector(10 downto 0);
60         counter_enable     : in    std_logic;
61         counter_clear_n    : in    std_logic;
62         counter_up_downN   : in    std_logic;
63         counter_out        : out   std_logic_vector(10 downto 0);
64         -- control signals
65         system_clear_n     : in    std_logic;
66         system_enable      : in    std_logic;
67         burstlen_enable    : in    std_logic;
68         burstlen_counter_enable : in    std_logic;
69         outreg_enable      : in    std_logic;
70         data_counter_enable : in    std_logic;
71         busy               : in    std_logic;
72         validout           : in    std_logic;
73         outpipe_enable     : in    std_logic;
74         outpipe_clear_n    : in    std_logic;
75         -- status signals
76         burst_end          : out   std_logic;
77         start_sampling     : out   std_logic;
78         enable             : out   std_logic;
79         clear_n            : out   std_logic
80     );

```



```

81  end component; -----
82
83  -- COMPONENT: control unit -----
84  component synchronizer_CU is
85  port
86  (
87    -- clock and reset
88    clk                : in    std_logic;
89    rst_n              : in    std_logic;
90    -- status signals
91    burst_end          : in    std_logic;
92    start_sampling     : in    std_logic;
93    enable              : in    std_logic;
94    clear_n            : in    std_logic;
95    -- control signals
96    system_clear_n     : out   std_logic;
97    system_enable      : out   std_logic;
98    burstlen_enable    : out   std_logic;
99    burstlen_counter_enable : out std_logic;
100   outreg_enable       : out   std_logic;
101   data_counter_enable : out   std_logic;
102   busy                : out   std_logic;
103   validout            : out   std_logic;
104   outpipe_enable      : out   std_logic;
105   outpipe_clear_n     : out   std_logic;
106 );
107 end component; -----
108
109 -- SIGNALS -----
110 signal burst_end          : std_logic;
111 signal start_sampling     : std_logic;
112 signal enable             : std_logic;
113 signal clear_n            : std_logic;
114 signal system_clear_n     : std_logic;
115 signal system_enable      : std_logic;
116 signal burstlen_enable    : std_logic;
117 signal burstlen_counter_enable : std_logic;
118 signal outreg_enable       : std_logic;
119 signal data_counter_enable : std_logic;
120 signal busy                : std_logic;
121 signal validout            : std_logic;
122 signal outpipe_enable      : std_logic;
123 signal outpipe_clear_n     : std_logic;
124 -----
125
126 begin
127
128   -- execution unit -----
129   EU: synchronizer_EU
130   port map
131   (
132     clk                => clk,
133     rst_n              => rst_n,
134     synch_enable        => synch_enable,
135     synch_clear_n       => synch_clear_n,
136     synch_strobe        => synch_strobe,
137     synch_validout      => synch_validout,
138     synch_busy          => synch_busy,
139     synch_din           => synch_din,
140     synch_dout          => synch_dout,
141     burstcount          => burstcount,
142     counter_enable      => counter_enable,
143     counter_clear_n     => counter_clear_n,
144     counter_up_downN    => counter_up_downN,
145     counter_out         => counter_out,
146     system_clear_n     => system_clear_n,
147     system_enable       => system_enable,
148     burstlen_enable     => burstlen_enable,
149     burstlen_counter_enable => burstlen_counter_enable,
150     outreg_enable       => outreg_enable,
151     data_counter_enable => data_counter_enable,
152     busy                => busy,
153     validout            => validout,
154     outpipe_enable      => outpipe_enable,
155     outpipe_clear_n     => outpipe_clear_n,
156     burst_end          => burst_end,

```

```

157     start_sampling      => start_sampling,
158     enable              => enable,
159     clear_n             => clear_n
160 ); -----
161
162 -- control unit -----
163 CU: synchronizer_CU
164 port map
165 (
166     clk                 => clk,
167     rst_n               => rst_n,
168     burst_end          => burst_end,
169     start_sampling     => start_sampling,
170     enable              => enable,
171     clear_n            => clear_n,
172     system_clear_n    => system_clear_n,
173     system_enable     => system_enable,
174     burstlen_enable   => burstlen_enable,
175     burstlen_counter_enable => burstlen_counter_enable,
176     outreg_enable     => outreg_enable,
177     data_counter_enable => data_counter_enable,
178     busy               => busy,
179     validout          => validout,
180     outpipe_enable    => outpipe_enable,
181     outpipe_clear_n   => outpipe_clear_n
182 ); -----
183
184 end rtl;

```

```

1 -- BRIEF DESCRIPTION: 16-bit SDR to 8-bit DDR (msb-first) converter
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5 use ieee.numeric_std.all;
6
7 entity SDR_to_DDR_converter is
8 port
9 (
10  clk       : in  std_logic;
11  load      : in  std_logic;
12  SDR_in    : in  std_logic_vector(15 downto 0);
13  DDR_out   : out std_logic_vector(7  downto 0)
14 );
15 end SDR_to_DDR_converter;
16
17 architecture rtl of SDR_to_DDR_converter is
18
19 -- COMPONENT: register -----
20 component reg is
21 generic
22 (
23     N : integer := 8
24 );
25 port
26 (
27     clk       : in  std_logic;
28     enable    : in  std_logic;
29     clear_n   : in  std_logic; -- synchronous clear, active low
30     reset_n   : in  std_logic; -- asynchronous clear, active low
31     din       : in  std_logic_vector(N-1 downto 0);
32     dout      : out std_logic_vector(N-1 downto 0) := (others => '0')
33 );
34 end component; -----
35
36 -- COMPONENT: multiplexer 2 to 1 -----
37 component mux_2to1 is
38 generic
39 (
40     N : integer := 1
41 );
42 port
43 (
44     din_0 : in  std_logic_vector((N-1) downto 0);
45     din_1 : in  std_logic_vector((N-1) downto 0);

```

```

46     sel      : in  std_logic;
47     dout     : out std_logic_vector((N-1) downto 0)
48 );
49 end component; -----
50
51 -- SIGNALS -----
52 signal reg_out      : std_logic_vector(15 downto 0);
53 signal outmux_in0   : std_logic_vector(7  downto 0);
54 signal outmux_in1   : std_logic_vector(7  downto 0);
55 -----
56
57 begin
58
59     -- input register -----
60     input_reg: reg
61     generic map
62     (
63         N => 16
64     )
65     port map
66     (
67         clk      => clk,
68         enable   => load,
69         clear_n  => '1',
70         reset_n  => '1',
71         din      => SDR_in,
72         dout     => reg_out
73     ); -----
74
75     outmux_in0 <= reg_out(7  downto 0);
76     outmux_in1 <= reg_out(15 downto 8);
77
78     -- clocked multiplexer -----
79     outmux: mux_2to1
80     generic map
81     (
82         N => 8
83     )
84     port map
85     (
86         din_0    => outmux_in0,
87         din_1    => outmux_in1,
88         sel      => clk,
89         dout     => DDR_out
90     ); -----
91
92 end rtl;

```

```

1  -- BRIEF DESCRIPTION: it generates the content of the hram configuration registers
2
3  library ieee;
4  use      ieee.std_logic_1164.all;
5  use      ieee.numeric_std.all;
6
7  entity conf_builder is
8  generic
9  (
10     hybrid_burst_enable      : std_logic := '1';
11     burst_lenght             : std_logic_vector(1 downto 0) := "11";
12     initial_latency          : std_logic_vector(3 downto 0) := "0001";
13     drive_strength           : std_logic_vector(2 downto 0) := "000";
14     distributed_refresh_interval : std_logic_vector(1 downto 0) := "10"
15 );
16 port
17 (
18     conf_virtual : in  std_logic_vector(1 downto 0);
19     conf0_real   : out std_logic_vector(15 downto 0);
20     conf1_real   : out std_logic_vector(15 downto 0)
21 );
22 end conf_builder;
23
24 architecture rtl of conf_builder is
25
26     begin

```

```

27
28 -- configuration register 0 -----
29 conf0_real(15)      <= not(conf_virtual(0));
30 conf0_real(14 downto 12) <= drive_strength;
31 conf0_real(11 downto 8)  <= "1111";
32 conf0_real(7  downto 4)  <= initial_latency;
33 conf0_real(3)          <= not(conf_virtual(1));
34 conf0_real(2)          <= hybrid_burst_enable;
35 conf0_real(1  downto 0)  <= burst_lenght;
36 -----
37
38 -- configuration register 1 -----
39 conf1_real(1  downto 0)  <= distributed_refresh_interval;
40 conf1_real(15 downto 2)  <= (others => '0');
41 -----
42
43 end rtl;

```

```

1 -- BRIEF DESCRIPTION: it generates CA starting from all the required parameters
2
3 library    ieee;
4 use        ieee.std_logic_1164.all;
5 use        ieee.numeric_std.all;
6
7 entity CA_builder is
8 port
9 (
10  read_writeN    : in  std_logic;
11  config_access  : in  std_logic;
12  address         : in  std_logic_vector(31 downto 0);
13  CA              : out std_logic_vector(47 downto 0)
14 );
15 end CA_builder;
16
17 architecture rtl of CA_builder is
18
19  signal up_address    : std_logic_vector(18 downto 0);
20  signal low_address   : std_logic_vector(2  downto 0);
21
22  signal CA0A : std_logic_vector(7  downto 0);
23  signal CA0B : std_logic_vector(7  downto 0);
24  signal CA1A : std_logic_vector(7  downto 0);
25  signal CA1B : std_logic_vector(7  downto 0);
26  signal CA2A : std_logic_vector(7  downto 0);
27  signal CA2B : std_logic_vector(7  downto 0);
28
29  begin
30
31    up_address <= address(21 downto 3);
32    low_address <= address(2  downto 0);
33
34    CA0A(7)      <= read_writeN;
35    CA0A(6)      <= config_access;
36    CA0A(5 downto 0) <= "000000";
37    CA0B(7  downto 3) <= "00000";
38    CA0B(2  downto 0) <= up_address(18 downto 16);
39    CA1A         <= up_address(15 downto 8);
40    CA1B         <= up_address(7  downto 0);
41    CA2A         <= "00000000";
42    CA2B(7  downto 3) <= "00000";
43    CA2B(2  downto 0) <= low_address;
44
45    CA(47 downto 40) <= CA0A;
46    CA(39 downto 32) <= CA0B;
47    CA(31 downto 24) <= CA1A;
48    CA(23 downto 16) <= CA1B;
49    CA(15 downto 8)  <= CA2A;
50    CA(7  downto 0)  <= CA2B;
51
52 end rtl;

```

```

1  -- BRIEF DESCRIPTION: it separates CA in CA1, CA2 and CA3
2
3  library    ieee;
4  use        ieee.std_logic_1164.all;
5  use        ieee.numeric_std.all;
6
7  entity CA_unpacker is
8  port
9  (
10     clk          : in  std_logic;
11     load         : in  std_logic;
12     sel          : in  std_logic_vector(1 downto 0);
13     CA_packed    : in  std_logic_vector(47 downto 0);
14     CA_unpacked  : out std_logic_vector(15 downto 0)
15 );
16 end CA_unpacker;
17
18 architecture rtl of CA_unpacker is
19
20     -- COMPONENT: register -----
21     component reg is
22     generic
23     (
24         N : integer := 8
25     );
26     port
27     (
28         clk          : in  std_logic;
29         enable       : in  std_logic;
30         clear_n      : in  std_logic; -- synchronous clear, active low
31         reset_n      : in  std_logic; -- asynchronous clear, active low
32         din          : in  std_logic_vector(N-1 downto 0);
33         dout         : out std_logic_vector(N-1 downto 0) := (others => '0')
34     );
35     end component; -----
36
37     -- COMPONENT: multiplexer 4 to 1 -----
38     component mux_4to1 is
39     generic
40     (
41         N : integer := 1
42     );
43     port
44     (
45         din_00      : in  std_logic_vector((N-1) downto 0);
46         din_01      : in  std_logic_vector((N-1) downto 0);
47         din_10      : in  std_logic_vector((N-1) downto 0);
48         din_11      : in  std_logic_vector((N-1) downto 0);
49         sel         : in  std_logic_vector(1 downto 0);
50         dout        : out std_logic_vector((N-1) downto 0)
51     );
52     end component; -----
53
54     -- SIGNALS -----
55     signal CAO_out   : std_logic_vector(15 downto 0);
56     signal CA1_out   : std_logic_vector(15 downto 0);
57     signal CA2_out   : std_logic_vector(15 downto 0);
58     -----
59
60     begin
61
62         -- CAO register -----
63         CAO : reg
64         generic map
65         (
66             N => 16
67         )
68         port map
69         (
70             clk      => clk,
71             enable   => load,
72             clear_n  => '1',
73             reset_n  => '1',
74             din      => CA_packed(47 downto 32),
75             dout     => CAO_out
76         ); -----

```

```

77
78 -- CA1 register -----
79 CA1 : reg
80 generic map
81 (
82   N => 16
83 )
84 port map
85 (
86   clk      => clk,
87   enable   => load,
88   clear_n  => '1',
89   reset_n  => '1',
90   din      => CA_packed(31 downto 16),
91   dout     => CA1_out
92 ); -----
93
94 -- CA1 register -----
95 CA2 : reg
96 generic map
97 (
98   N => 16
99 )
100 port map
101 (
102   clk      => clk,
103   enable   => load,
104   clear_n  => '1',
105   reset_n  => '1',
106   din      => CA_packed(15 downto 0),
107   dout     => CA2_out
108 ); -----
109
110 -- output multiplexer -----
111 CA_mux : mux_4to1
112 generic map
113 (
114   N => 16
115 )
116 port map
117 (
118   din_00  => CA0_out,
119   din_01  => CA1_out,
120   din_10  => CA2_out,
121   din_11  => (others => '0'),
122   sel     => sel,
123   dout    => CA_unpacked
124 ); -----
125
126 end rtl;

```

```

1 -- BRIEF DESCRIPTION: execution unit of the AvalonMM - hyperram converter
2
3 library ieee;
4 use ieee.std_logic_1164.all;
5 use ieee.numeric_std.all;
6
7 entity avs_hram_mainconv_EU is
8 generic
9 (
10  drive_strength : std_logic_vector(2 downto 0) := "000"
11 );
12 port
13 (
14  clk           : in    std_logic;
15  reset_n      : in    std_logic;
16  -- avs signals
17  avs_address   : in    std_logic_vector(22 downto 0);
18  avs_read      : in    std_logic;
19  avs_readdata  : out   std_logic_vector(15 downto 0);
20  avs_write     : in    std_logic;
21  avs_writedata : in    std_logic_vector(15 downto 0);
22  avs_waitrequest : out  std_logic;
23  avs_readdatavalid : out std_logic;

```

```

24  avs_burstcount      : in      std_logic_vector(10 downto 0);
25  -- hram signals
26  hram_RESET_n       : out     std_logic;
27  hram_DQ             : inout   std_logic_vector(7  downto 0);
28  hram_RWDS          : inout   std_logic;
29  hram_CS_n          : out     std_logic;
30  clk90              : out     std_logic;
31  hCK_enable         : out     std_logic;
32  -- control signals
33  waitrequest        : in      std_logic;
34  force_valid        : in      std_logic;
35  cmd_load           : in      std_logic;
36  dpd_req_clear_n   : in      std_logic;
37  synch_cnt_enable   : in      std_logic;
38  synch_cnt_down     : in      std_logic;
39  synch_cnt_clear_n : in      std_logic;
40  datain_load       : in      std_logic;
41  avs_out_sel       : in      std_logic;
42  reset_config_register_n : in  std_logic;
43  update_config_register : in  std_logic;
44  address_space_sel : in      std_logic_vector(1  downto 0);
45  config_access     : in      std_logic;
46  read_writeN      : in      std_logic;
47  CA_load           : in      std_logic;
48  CA_sel            : in      std_logic_vector(1  downto 0);
49  dq_sel            : in      std_logic_vector(1  downto 0);
50  dq_OE             : in      std_logic;
51  writedata_load    : in      std_logic;
52  addressgen_enable : in      std_logic;
53  synch_enable      : in      std_logic;
54  synch_clear_n    : in      std_logic;
55  RWDS_sampling_enable : in  std_logic;
56  check_latency     : in      std_logic;
57  force_RWDS_low   : in      std_logic;
58  hCK_gating_enable_n : in  std_logic;
59  set_dpd_status    : in      std_logic;
60  clear_dpd_status_n : in  std_logic;
61  deadline_tim_enable : in  std_logic;
62  deadline_tim_clear_n : in  std_logic;
63  hCKen_pipe_clear_n : in  std_logic;
64  hbus_RESET_n     : in      std_logic;
65  hbus_CS_n        : in      std_logic;
66  -- status signals
67  write             : out     std_logic;
68  read              : out     std_logic;
69  config            : out     std_logic;
70  dpd_req           : out     std_logic;
71  active_dpd_req    : out     std_logic;
72  current_operation : out     std_logic;
73  bursttransfer     : out     std_logic;
74  burst_end        : out     std_logic;
75  synch_busy       : out     std_logic;
76  doubled_latency   : out     std_logic;
77  dpd_mode_on      : out     std_logic;
78  t_acc1           : out     std_logic;
79  t_acc2           : out     std_logic;
80  t_dpdcs1         : out     std_logic;
81  t_dpdin          : out     std_logic;
82  t_dpdout         : out     std_logic
83 );
84 end avs_hram_mainconv_EU;
85
86 architecture rtl of avs_hram_mainconv_EU is
87
88  -- CONSTANTS -----
89  constant hybrid_burst_enable      : std_logic := '1';
90  constant burst_lenght             : std_logic_vector(1  downto 0) := "11";
91  constant initial_latency          : std_logic_vector(3  downto 0) := "1111";
92  constant distributed_refresh_interval : std_logic_vector(1  downto 0) := "10";
93  -----
94
95  -- COMPONENT: flipflop set-reset -----
96  component sr_flipflop is
97  port
98  (
99    clk      : in std_logic;

```

```

100     set      : in std_logic;
101     clear_n  : in std_logic;
102     rst_n    : in std_logic;
103     dout     : out std_logic
104 );
105 end component; -----
106
107 -- COMPONENT: flipflop type D -----
108 component d_flipflop is
109 port
110 (
111     clk      : in  std_logic;
112     enable   : in  std_logic;
113     clear_n  : in  std_logic; -- synchronous clear, active low
114     reset_n  : in  std_logic; -- asynchronous reset, active low
115     din      : in  std_logic;
116     dout     : out std_logic
117 );
118 end component; -----
119
120 -- COMPONENT: register -----
121 component reg is
122 generic
123 (
124     N : integer := 8
125 );
126 port
127 (
128     clk      : in  std_logic;
129     enable   : in  std_logic;
130     clear_n  : in  std_logic; -- synchronous clear, active low
131     reset_n  : in  std_logic; -- asynchronous clear, active low
132     din      : in  std_logic_vector(N-1 downto 0);
133     dout     : out std_logic_vector(N-1 downto 0) := (others => '0')
134 );
135 end component; -----
136
137 -- COMPONENT: multiplexer 2-inputs 1-output -----
138 component mux_2to1 is
139 generic
140 (
141     N : integer := 1
142 );
143 port
144 (
145     din_0    : in  std_logic_vector((N-1) downto 0);
146     din_1    : in  std_logic_vector((N-1) downto 0);
147     sel      : in  std_logic;
148     dout     : out std_logic_vector((N-1) downto 0)
149 );
150 end component; -----
151
152 -- COMPONENT: multiplexer 4-inputs 1-output -----
153 component mux_4to1 is
154 generic
155 (
156     N : integer := 1
157 );
158 port
159 (
160     din_00   : in  std_logic_vector((N-1) downto 0);
161     din_01   : in  std_logic_vector((N-1) downto 0);
162     din_10   : in  std_logic_vector((N-1) downto 0);
163     din_11   : in  std_logic_vector((N-1) downto 0);
164     sel      : in  std_logic_vector(1 downto 0);
165     dout     : out std_logic_vector((N-1) downto 0)
166 );
167 end component; -----
168
169 -- COMPONENT: comparator -----
170 component comparator_Nbit is
171 generic
172 (
173     N : integer := 1
174 );
175 port

```



```

176 (
177     din_0    : in    std_logic_vector((N-1) downto 0);
178     din_1    : in    std_logic_vector((N-1) downto 0);
179     equal    : out   std_logic
180 );
181 end component; -----
182
183 -- COMPONENT: timer -----
184 component timer_14bit is
185 port
186 (
187     clk       : in    std_logic;
188     enable    : in    std_logic;
189     clear_n   : in    std_logic;
190     tim_3     : out   std_logic;
191     tim_7     : out   std_logic;
192     tim_21    : out   std_logic;
193     tim_1000  : out   std_logic;
194     tim_15000 : out   std_logic
195 );
196 end component; -----
197
198 -- COMPONENT: DDR to SDR converter -----
199 component DDR_to_SDR_converter is
200 port
201 (
202     -- clock, reset and clear
203     clk_x8      : in    std_logic;
204     clear_n     : in    std_logic;
205     -- IO signals
206     rwds_in     : in    std_logic;
207     rwds_out    : out   std_logic;
208     DDR_in      : in    std_logic_vector(7 downto 0);
209     SDR_out     : out   std_logic_vector(15 downto 0)
210 );
211 end component; -----
212
213 -- COMPONENT: SDR to DDR converter -----
214 component SDR_to_DDR_converter is
215 port
216 (
217     clk       : in    std_logic;
218     load      : in    std_logic;
219     SDR_in    : in    std_logic_vector(15 downto 0);
220     DDR_out   : out   std_logic_vector(7 downto 0)
221 );
222 end component; -----
223
224 -- COMPONENT: CA builder -----
225 component CA_builder is
226 port
227 (
228     read_writeN : in    std_logic;
229     config_access : in    std_logic;
230     address      : in    std_logic_vector(31 downto 0);
231     CA           : out   std_logic_vector(47 downto 0)
232 );
233 end component; -----
234
235 -- COMPONENT: CA unpacker -----
236 component CA_unpacker is
237 port
238 (
239     clk       : in    std_logic;
240     load      : in    std_logic;
241     sel       : in    std_logic_vector(1 downto 0);
242     CA_packed : in    std_logic_vector(47 downto 0);
243     CA_unpacked : out   std_logic_vector(15 downto 0)
244 );
245 end component; -----
246
247 -- COMPONENT: configuration registers builder -----
248 component conf_builder is
249 generic
250 (
251     hybrid_burst_enable : std_logic := '1';

```

```

252     burst_lenght      : std_logic_vector(1 downto 0) := "11";
253     initial_latency  : std_logic_vector(3 downto 0) := "0001";
254     drive_strength   : std_logic_vector(2 downto 0) := "000";
255     distributed_refresh_interval : std_logic_vector(1 downto 0) := "10"
256 );
257 port
258 (
259     conf_virtual : in  std_logic_vector(1 downto 0);
260     conf0_real   : out std_logic_vector(15 downto 0);
261     conf1_real   : out std_logic_vector(15 downto 0)
262 );
263 end component; -----
264
265 -- COMPONENT: synchronizer -----
266 component synchronizer is
267 port
268 (
269     clk           : in  std_logic;
270     rst_n         : in  std_logic;
271     synch_enable  : in  std_logic;
272     synch_clear_n : in  std_logic;
273     synch_strobe  : in  std_logic;
274     synch_validout : out std_logic;
275     synch_busy    : out std_logic;
276     synch_din     : in  std_logic_vector(15 downto 0);
277     synch_dout    : out std_logic_vector(15 downto 0);
278     burstcount    : in  std_logic_vector(10 downto 0);
279     counter_enable : in  std_logic;
280     counter_clear_n : in  std_logic;
281     counter_up_downN : in  std_logic;
282     counter_out    : out std_logic_vector(10 downto 0)
283 );
284 end component; -----
285
286 -- COMPONENT: tristate buffer -----
287 component tristate_buffer is
288 generic
289 (
290     N : integer := 8
291 );
292 port
293 (
294     enable : in  std_logic;
295     din    : in  std_logic_vector(N-1 downto 0);
296     dout   : out std_logic_vector(N-1 downto 0)
297 );
298 end component; -----
299
300 -- COMPONENT: DLL with 90 degree delay -----
301 component dll_90 is
302 port
303 (
304     areset : in  std_logic;
305     inclk0 : in  std_logic;
306     c0     : out std_logic
307 );
308 end component; -----
309
310 -- COMPONENT: PLL from 50 MHz to 400 MHz -----
311 component pll_x8 is
312 port
313 (
314     areset : in  std_logic := '0';
315     inclk0 : in  std_logic := '0';
316     c0     : out std_logic
317 );
318 end component; -----
319
320 -- COMPONENT: adder 22-bit 1-pipe -----
321 component adder_22bit_1pipe is
322 port
323 (
324     clken : in  std_logic ;
325     clock : in  std_logic ;
326     dataa : in  std_logic_vector (21 downto 0);
327     datab : in  std_logic_vector (21 downto 0);

```

```

328     result : out std_logic_vector (21 downto 0)
329 );
330 end component; -----
331
332 -- CONSTANTS -----
333 constant config0_addr: std_logic_vector(31 downto 0) := "00000000000000000000100000000000";
334 constant config1_addr: std_logic_vector(31 downto 0) := "00000000000000000000100000000001";
335 -----
336
337 -- SIGNALS -----
338 signal pll_reset           : std_logic;
339 signal synch_cnt_up_downN : std_logic;
340 signal burst_detector_out : std_logic;
341 signal synch_validout     : std_logic;
342 signal conf_reg_out       : std_logic_vector(1 downto 0);
343 signal burstcnt_reg_out   : std_logic_vector(10 downto 0);
344 signal addr_reg_out       : std_logic_vector(22 downto 0);
345 signal extended_address   : std_logic_vector(31 downto 0);
346 signal datain_reg_out     : std_logic_vector(15 downto 0);
347 signal conf0_real         : std_logic_vector(15 downto 0);
348 signal conf1_real         : std_logic_vector(15 downto 0);
349 signal dq_mux_out         : std_logic_vector(15 downto 0);
350 signal CA_unpacked        : std_logic_vector(15 downto 0);
351 signal addressgen_out     : std_logic_vector(21 downto 0);
352 signal generated_address  : std_logic_vector(31 downto 0);
353 signal effective_address  : std_logic_vector(31 downto 0);
354 signal CA                  : std_logic_vector(47 downto 0);
355 signal readdata_SDR       : std_logic_vector(15 downto 0);
356 signal synch_dout         : std_logic_vector(15 downto 0);
357 signal synch_cnt_out      : std_logic_vector(10 downto 0);
358 signal readdatamux_din0   : std_logic_vector(15 downto 0);
359 signal cntpipe1_out       : std_logic_vector(10 downto 0);
360 signal cntpipe2_out       : std_logic_vector(10 downto 0);
361 signal cntpipe3_out       : std_logic_vector(10 downto 0);
362 signal dataa              : std_logic_vector(21 downto 0);
363 signal writedata_conv_out  : std_logic_vector(7 downto 0);
364 signal RWDS_buffer_out    : std_logic_vector(0 downto 0);
365 signal shifted_RWDS       : std_logic;
366 signal clk_x8              : std_logic;
367 -----
368
369 begin
370
371     hram_CS_n <= hbus_CS_n;
372     hram_RESET_n <= hbus_RESET_n;
373     avs_waitrequest <= waitrequest;
374     read <= avs_read;
375     write <= avs_write;
376     dpd_req <= avs_writedata(0);
377
378     -- virtual configuration register access -----
379     config <=
380         (    avs_address(22))    and (not avs_address(21))    and (not avs_address(20))    and
381         (not avs_address(19))    and (not avs_address(18))    and (not avs_address(17))    and
382         (not avs_address(16))    and (not avs_address(15))    and (not avs_address(14))    and
383         (not avs_address(13))    and (not avs_address(12))    and (not avs_address(11))    and
384         (not avs_address(10))    and (not avs_address(9 ))     and (not avs_address(8 ))     and
385         (not avs_address(7 ))    and (not avs_address(6 ))     and (not avs_address(5 ))     and
386         (not avs_address(4 ))    and (not avs_address(3 ))     and (not avs_address(2 ))     and
387         (not avs_address(1 ))    and (not avs_address(0 ))     ;
388     -----
389
390     -- address register -----
391     addr_reg : reg
392     generic map
393     (
394         N => 23
395     )
396     port map
397     (
398         clk      => clk,
399         enable   => cmd_load,
400         clear_n  => '1',
401         reset_n  => '1',
402         din      => avs_address,
403         dout     => addr_reg_out

```

```

404 ); -----
405
406 -- burstcount register -----
407 burstcnt_reg : reg
408 generic map
409 (
410   N => 11
411 )
412 port map
413 (
414   clk      => clk,
415   enable   => cmd_load,
416   clear_n  => '1',
417   reset_n  => '1',
418   din      => avs_burstcount,
419   dout     => burstcnt_reg_out
420 ); -----
421
422 -- datain register -----
423 datain_reg : reg
424 generic map
425 (
426   N => 16
427 )
428 port map
429 (
430   clk      => clk,
431   enable   => datain_load,
432   clear_n  => '1',
433   reset_n  => '1',
434   din      => avs_writedata,
435   dout     => datain_reg_out
436 ); -----
437
438 -- virtual configuration register -----
439 conf_reg : reg
440 generic map
441 (
442   N => 2
443 )
444 port map
445 (
446   clk      => clk,
447   enable   => update_config_register,
448   clear_n  => reset_config_register_n,
449   reset_n  => '1',
450   din      => datain_reg_out(1 downto 0),
451   dout     => conf_reg_out
452 ); -----
453
454 -- configuration registers builder -----
455 conf_builder_inst : conf_builder
456 generic map
457 (
458   hybrid_burst_enable      => hybrid_burst_enable,
459   burst_lenght             => burst_lenght,
460   initial_latency          => initial_latency,
461   drive_strength           => drive_strength,
462   distributed_refresh_interval => distributed_refresh_interval
463 )
464 port map
465 (
466   conf_virtual => conf_reg_out,
467   conf0_real  => conf0_real,
468   conf1_real  => conf1_real
469 ); -----
470
471 -- DQ selection -----
472 dq_mux : mux_4to1
473 generic map
474 (
475   N => 16
476 )
477 port map
478 (
479   din_00      => datain_reg_out,

```

```

480     din_01    => conf0_real,
481     din_10    => conf1_real,
482     din_11    => CA_unpacked,
483     sel       => dq_sel,
484     dout      => dq_mux_out
485 ); -----
486
487 extended_address(22 downto 0) <= addr_reg_out;
488 extended_address(31 downto 23) <= (others => '0');
489
490 -- address space selection -----
491 address_mux : mux_4to1
492 generic map
493 (
494     N => 32
495 )
496 port map
497 (
498     din_00    => extended_address,
499     din_01    => config0_addr,
500     din_10    => config1_addr,
501     din_11    => generated_address,
502     sel       => address_space_sel,
503     dout      => effective_address
504 ); -----
505
506 -- CA builder -----
507 CA_builder_inst : CA_builder
508 port map
509 (
510     read_writeN    => read_writeN,
511     config_access  => config_access,
512     address        => effective_address,
513     CA             => CA
514 ); -----
515
516 -- CA unpacker -----
517 CA_unpacker_inst : CA_unpacker
518 port map
519 (
520     clk          => clk,
521     load         => CA_load,
522     sel         => CA_sel,
523     CA_packed   => CA,
524     CA_unpacked => CA_unpacked
525 ); -----
526
527 -- DQ tristate buffer -----
528 dq_buffer : tristate_buffer
529 generic map
530 (
531     N => 8
532 )
533 port map
534 (
535     enable    => dq_OE,
536     din       => writedata_conv_out,
537     dout      => hram_DQ
538 ); -----
539
540 -- writedata converter (SDR to DDR) -----
541 writedata_converter : SDR_to_DDR_converter
542 port map
543 (
544     clk        => clk,
545     load       => writedata_load,
546     SDR_in    => dq_mux_out,
547     DDR_out   => writedata_conv_out
548 ); -----
549
550 pll_reset <= not reset_n;
551
552 -- clock shifter -----
553 clk_shifter: dll_90
554 port map
555 (

```

```

556     areset    => pll_reset,
557     inclk0   => clk,
558     c0       => clk90
559 ); -----
560
561 -- gating enable pipeline register -----
562 hCKen_pipe : d_flipflop
563 port map
564 (
565     clk       => clk,
566     enable    => '1',
567     clear_n   => hCKen_pipe_clear_n,
568     reset_n   => '1',
569     din       => hCK_gating_enable_n,
570     dout      => hCK_enable
571 ); -----
572
573 -- RWDS tristate buffer -----
574 RWDS_buffer : tristate_buffer
575 generic map
576 (
577     N => 1
578 )
579 port map
580 (
581     enable    => force_RWDS_low,
582     din       => (others => '0'),
583     dout      => RWDS_buffer_out
584 ); -----
585
586 hram_RWDS <= RWDS_buffer_out(0);
587
588 -- RWDS tracker -----
589 rwds_tracker : d_flipflop
590 port map
591 (
592     clk       => clk,
593     enable    => check_latency,
594     clear_n   => '1',
595     reset_n   => '1',
596     din       => hram_RWDS,
597     dout      => doubled_latency
598 ); -----
599
600 -- pll from 50 MHz to 400 MHz -----
601 pll_x8_inst: pll_x8
602 port map
603 (
604     areset    => pll_reset,
605     inclk0   => clk,
606     c0       => clk_x8
607 ); -----
608
609 -- readdata converter (DDR to SDR) -----
610 readdata_converter : DDR_to_SDR_converter
611 port map
612 (
613     clk_x8    => clk_x8,
614     clear_n   => RWDS_sampling_enable,
615     rwds_in   => hram_RWDS,
616     rwds_out  => shifted_RWDS,
617     DDR_in    => hram_DQ,
618     SDR_out   => readdata_SDR
619 ); -----
620
621 synch_cnt_up_downN <= not synch_cnt_down;
622
623 -- synchronizer -----
624 synchronizer_inst : synchronizer
625 port map
626 (
627     clk       => clk,
628     rst_n     => reset_n,
629     synch_enable => synch_enable,
630     synch_clear_n => synch_clear_n,
631     synch_strobe => shifted_RWDS,

```

```

632     synch_validout    => synch_validout ,
633     synch_busy       => synch_busy ,
634     synch_din        => readdata_SDR ,
635     synch_dout        => synch_dout ,
636     burstcount       => burstcnt_reg_out ,
637     counter_enable    => synch_cnt_enable ,
638     counter_clear_n   => synch_cnt_clear_n ,
639     counter_up_downN  => synch_cnt_up_downN ,
640     counter_out       => synch_cnt_out
641 ); -----
642
643 avs_readdatavalid <= synch_validout or force_valid;
644
645 readdatamux_din0(1 downto 0) <= conf_reg_out;
646 readdatamux_din0(15 downto 2) <= (others => '0');
647
648 -- readdata selection -----
649 readdatamux : mux_2to1
650 generic map
651 (
652     N => 16
653 )
654 port map
655 (
656     din_0    => synch_dout ,
657     din_1    => readdatamux_din0 ,
658     sel      => avs_out_sel ,
659     dout     => avs_readdata
660 ); -----
661
662 -- counter pipeline register 1 -----
663 cntpipe1 : reg
664 generic map
665 (
666     N => 11
667 )
668 port map
669 (
670     clk      => clk ,
671     enable   => '1' ,
672     clear_n  => '1' ,
673     reset_n  => '1' ,
674     din      => synch_cnt_out ,
675     dout     => cntpipe1_out
676 ); -----
677
678 -- counter pipeline register 2 -----
679 cntpipe2 : reg
680 generic map
681 (
682     N => 11
683 )
684 port map
685 (
686     clk      => clk ,
687     enable   => '1' ,
688     clear_n  => '1' ,
689     reset_n  => '1' ,
690     din      => cntpipe1_out ,
691     dout     => cntpipe2_out
692 ); -----
693
694 -- counter pipeline register 3 -----
695 cntpipe3 : reg
696 generic map
697 (
698     N => 11
699 )
700 port map
701 (
702     clk      => clk ,
703     enable   => '1' ,
704     clear_n  => '1' ,
705     reset_n  => '1' ,
706     din      => cntpipe2_out ,
707     dout     => cntpipe3_out

```

```

708 ); -----
709
710 dataa(10 downto 0) <= cntpipe3_out;
711 dataa(21 downto 11) <= (others => '0');
712
713 -- address generator for burst recovery -----
714 addressgen : adder_22bit_1pipe
715 port map
716 (
717     clken    => addressgen_enable,
718     clock    => clk,
719     dataa    => dataa,
720     datab   => addr_reg_out(21 downto 0),
721     result  => addressgen_out
722 ); -----
723
724 generated_address(21 downto 0) <= addressgen_out;
725 generated_address(31 downto 22) <= (others => '0');
726
727 -- burst length comparator -----
728 burst_cmp : comparator_Nbit
729 generic map
730 (
731     N => 11
732 )
733 port map
734 (
735     din_0    => burstcnt_reg_out,
736     din_1    => synch_cnt_out,
737     equal    => burst_end
738 ); -----
739
740 -- burst detector -----
741 burst_detector : comparator_Nbit
742 generic map
743 (
744     N => 11
745 )
746 port map
747 (
748     din_0    => burstcnt_reg_out,
749     din_1    => "0000000001",
750     equal    => burst_detector_out
751 ); -----
752
753 bursttransfer <= not burst_detector_out;
754
755 -- DPD request tracker -----
756 dpd_req_tracker : d_flipflop
757 port map
758 (
759     clk       => clk,
760     enable    => cmd_load,
761     clear_n   => dpd_req_clear_n,
762     reset_n   => '1',
763     din       => avs_writedata(0),
764     dout      => active_dpd_req
765 ); -----
766
767 -- operation type tracker -----
768 op_tracker : d_flipflop
769 port map
770 (
771     clk       => clk,
772     enable    => cmd_load,
773     clear_n   => '1',
774     reset_n   => '1',
775     din       => avs_write,
776     dout      => current_operation
777 ); -----
778
779 -- DPD mode tracker -----
780 dpd_tracker : sr_flipflop
781 port map
782 (
783     clk       => clk,

```



```

784     set      => set_dpd_status,
785     clear_n => clear_dpd_status_n,
786     rst_n   => '1',
787     dout    => dpd_mode_on
788 ); -----
789
790 -- deadline timer -----
791 deadline_timer : timer_14bit
792 port map
793 (
794     clk      => clk,
795     enable   => deadline_tim_enable,
796     clear_n  => deadline_tim_clear_n,
797     tim_3    => t_acc1,
798     tim_7    => t_acc2,
799     tim_21   => t_dpdcsl,
800     tim_1000 => t_dpdin,
801     tim_15000 => t_dp dout
802 ); -----
803
804 end rtl;

```

```

1  -- BRIEF DESCRIPTION: control unit of the AvalonMM - hyperram converter
2
3  library ieee;
4  use      ieee.std_logic_1164.all;
5  use      ieee.numeric_std.all;
6
7  entity avs_hram_mainconv_CU is
8  port
9  (
10     clk                : in  std_logic;
11     reset_n            : in  std_logic;
12     -- control signals
13     waitrequest        : out std_logic;
14     force_valid        : out std_logic;
15     cmd_load           : out std_logic;
16     dpd_req_clear_n   : out std_logic;
17     synch_cnt_enable   : out std_logic;
18     synch_cnt_down    : out std_logic;
19     synch_cnt_clear_n : out std_logic;
20     datain_load        : out std_logic;
21     avs_out_sel        : out std_logic;
22     reset_config_register_n : out std_logic;
23     update_config_register : out std_logic;
24     address_space_sel  : out std_logic_vector(1 downto 0);
25     config_access      : out std_logic;
26     read_writeN        : out std_logic;
27     CA_load            : out std_logic;
28     CA_sel             : out std_logic_vector(1 downto 0);
29     dq_sel            : out std_logic_vector(1 downto 0);
30     dq_OE             : out std_logic;
31     writedata_load    : out std_logic;
32     addressgen_enable  : out std_logic;
33     synch_enable      : out std_logic;
34     synch_clear_n     : out std_logic;
35     RWDS_sampling_enable : out std_logic;
36     check_latency     : out std_logic;
37     force_RWDS_low    : out std_logic;
38     hCK_gating_enable_n : out std_logic;
39     set_dpd_status    : out std_logic;
40     clear_dpd_status_n : out std_logic;
41     deadline_tim_enable : out std_logic;
42     deadline_tim_clear_n : out std_logic;
43     hCKen_pipe_clear_n : out std_logic;
44     hbus_RESET_n      : out std_logic;
45     hbus_CS_n         : out std_logic;
46     -- status signals
47     write              : in  std_logic;
48     read               : in  std_logic;
49     config             : in  std_logic;
50     dpd_req            : in  std_logic;
51     active_dpd_req    : in  std_logic;
52     current_operation  : in  std_logic;

```

```

53 bursttransfer      : in  std_logic;
54 burst_end         : in  std_logic;
55 synch_busy        : in  std_logic;
56 doubled_latency   : in  std_logic;
57 dpd_mode_on       : in  std_logic;
58 t_acc1            : in  std_logic;
59 t_acc2            : in  std_logic;
60 t_dpdcsl          : in  std_logic;
61 t_dpdin           : in  std_logic;
62 t_dp dout         : in  std_logic
63 );
64 end entity av_s_hram_mainconv_CU;
65
66 architecture fsm of av_s_hram_mainconv_CU is
67
68   -- states definition -----
69   type state is
70   (
71     reset,
72     reset_wait,
73     reset_exit_begin,
74     reset_exit,
75     idle,
76     dummycmd,
77     dummycmd_last,
78     dummycmd_end,
79     wait_dpd_out,
80     write_virtconf,
81     writeconf0_prep,
82     writeconf_CA0,
83     writeconf_CA1,
84     writeconf_CA2,
85     writeconf0,
86     writeconf0_end,
87     wait_dpd_in,
88     read_virtconf,
89     readmem_prep,
90     writemem_prep,
91     CA_0,
92     CA_1,
93     CA_2,
94     CA_end,
95     read_wait_0,
96     read_wait_1,
97     read_wait_2,
98     read_end_1,
99     read_end_2,
100    synch_restoring_1,
101    synch_restoring_2,
102    writemem_wait,
103    writemem,
104    write_end,
105    writeburst_prep,
106    writeburst,
107    writeburst_last,
108    stop_burst_1,
109    stop_burst_2,
110    idle_burst,
111    restore_burst
112 ); -----
113
114   -- states declaration -----
115   signal present_state : state;
116   signal next_state    : state;
117   -----
118
119   begin
120
121     -- evaluation of the next state -----
122     next_state_evaluation: process
123     (
124       -- sensitivity list
125       present_state,
126       write,
127       read,
128       config,

```

```

129     dpd_req,
130     active_dpd_req,
131     current_operation,
132     bursttransfer,
133     burst_end,
134     synch_busy,
135     doubled_latency,
136     dpd_mode_on,
137     t_acc1,
138     t_acc2,
139     t_dpdcs1,
140     t_dpdin,
141     t_dpdout
142 )
143 begin
144     case present_state is
145         -----
146         when reset =>
147             next_state <= reset_wait;
148         -----
149         when reset_wait =>
150             if (t_dpdcs1 = '1') then
151                 next_state <= reset_exit_begin;
152             else
153                 next_state <= reset_wait;
154             end if;
155         -----
156         when reset_exit_begin =>
157             next_state <= reset_exit;
158         -----
159         when reset_exit =>
160             if (t_dpdout = '1') then
161                 next_state <= writeconf0_prep;
162             else
163                 next_state <= reset_exit;
164             end if;
165         -----
166         when idle | read_virtconf =>
167             if (read = '1') then
168                 if (config = '1') then
169                     next_state <= read_virtconf;
170                 else
171                     if (dpd_mode_on = '1') then
172                         next_state <= idle;
173                     else
174                         next_state <= readmem_prep;
175                     end if;
176                 end if;
177             elsif (write = '1') then
178                 if (config = '1') then
179                     if (dpd_mode_on = '1') then
180                         if (dpd_req = '1') then
181                             next_state <= idle;
182                         else
183                             next_state <= dummycmd;
184                         end if;
185                     else
186                         next_state <= write_virtconf;
187                     end if;
188                 else
189                     if (dpd_mode_on = '1') then
190                         next_state <= idle;
191                     else
192                         next_state <= writemem_prep;
193                     end if;
194                 end if;
195             else
196                 next_state <= idle;
197             end if;
198         -----
199         when dummycmd =>
200             if (t_dpdcs1 = '1') then
201                 next_state <= dummycmd_last;
202             else
203                 next_state <= dummycmd;
204             end if;

```

```

205 -----
206 when dummycmd_last =>
207     next_state <= dummycmd_end;
208 -----
209 when dummycmd_end =>
210     next_state <= wait_dpd_out;
211 -----
212 when wait_dpd_out =>
213     if (t_dpdout = '1') then
214         next_state <= write_virtconf;
215     else
216         next_state <= wait_dpd_out;
217     end if;
218 -----
219 when write_virtconf =>
220     next_state <= writeconf0_prep;
221 -----
222 when writeconf0_prep =>
223     next_state <= writeconf_CA0;
224 -----
225 when writeconf_CA0 =>
226     next_state <= writeconf_CA1;
227 -----
228 when writeconf_CA1 =>
229     next_state <= writeconf_CA2;
230 -----
231 when writeconf_CA2 =>
232     next_state <= writeconf0;
233 -----
234 when writeconf0 =>
235     next_state <= writeconf0_end;
236 -----
237 when writeconf0_end =>
238     if (active_dpd_req = '1') then
239         next_state <= wait_dpd_in;
240     else
241         next_state <= idle;
242     end if;
243 -----
244 when wait_dpd_in =>
245     if (t_dpdin = '1') then
246         next_state <= idle;
247     else
248         next_state <= wait_dpd_in;
249     end if;
250 -----
251 when readmem_prep | writemem_prep | restore_burst =>
252     next_state <= CA_0;
253 -----
254 when CA_0 =>
255     next_state <= CA_1;
256 -----
257 when CA_1 =>
258     next_state <= CA_2;
259 -----
260 when CA_2 =>
261     next_state <= CA_end;
262 -----
263 when CA_end =>
264     if (current_operation = '1') then
265         next_state <= writemem_wait;
266     else
267         next_state <= read_wait_0;
268     end if;
269 -----
270 when read_wait_0 =>
271     next_state <= read_wait_1;
272 -----
273 when read_wait_1 =>
274     if (synch_busy = '1') then
275         next_state <= read_wait_2;
276     else
277         next_state <= read_wait_1;
278     end if;
279 -----
280 when read_wait_2 =>

```

```

281     if (synch_busy = '1') then
282         next_state <= read_wait_2;
283     else
284         next_state <= read_end_1;
285     end if;
286 -----
287 when read_end_1 =>
288     next_state <= read_end_2;
289 -----
290 when read_end_2 =>
291     next_state <= synch_restoring_1;
292 -----
293 when synch_restoring_1 =>
294     next_state <= synch_restoring_2;
295 -----
296 when synch_restoring_2 =>
297     next_state <= idle;
298 -----
299 when writemem_wait =>
300     if (doubled_latency = '1') then
301         if (t_acc2 = '1') then
302             if (bursttransfer = '1') then
303                 next_state <= writeburst_prep;
304             else
305                 next_state <= writemem;
306             end if;
307         else
308             next_state <= writemem_wait;
309         end if;
310     else
311         if (t_acc1 = '1') then
312             if (bursttransfer = '1') then
313                 next_state <= writeburst_prep;
314             else
315                 next_state <= writemem;
316             end if;
317         else
318             next_state <= writemem_wait;
319         end if;
320     end if;
321 -----
322 when writemem | writeburst_last =>
323     next_state <= write_end;
324 -----
325 when write_end =>
326     next_state <= idle;
327 -----
328 when writeburst_prep | writeburst =>
329     if (write = '1') then
330         if (burst_end = '1') then
331             next_state <= writeburst_last;
332         else
333             next_state <= writeburst;
334         end if;
335     else
336         next_state <= stop_burst_1;
337     end if;
338 -----
339 when stop_burst_1 =>
340     next_state <= stop_burst_2;
341 -----
342 when stop_burst_2 =>
343     next_state <= idle_burst;
344 -----
345 when idle_burst =>
346     if (write = '1') then
347         next_state <= restore_burst;
348     else
349         next_state <= idle_burst;
350     end if;
351 -----
352 when others =>
353     next_state <= reset;
354 -----
355 end case;
356 end process next_state_evaluation; -----

```

```

357
358 -- state transition -----
359 state_transition: process (clk, reset_n)
360 begin
361     if (reset_n = '0') then
362         present_state <= reset;
363     elsif (rising_edge(clk)) then
364         present_state <= next_state;
365     end if;
366 end process state_transition; -----
367
368 -- control signals definition -----
369 control_signals_definition: process (present_state)
370 begin
371     -- default values -----
372     waitrequest                <= '0';
373     force_valid                <= '0';
374     cmd_load                   <= '0';
375     dpd_req_clear_n           <= '1';
376     synch_cnt_enable          <= '0';
377     synch_cnt_down            <= '0';
378     synch_cnt_clear_n         <= '1';
379     datain_load               <= '0';
380     avs_out_sel               <= '0';
381     reset_config_register_n   <= '0';
382     update_config_register    <= '0';
383     address_space_sel         <= "00";
384     config_access             <= '0';
385     read_writeN               <= '0';
386     CA_load                   <= '0';
387     CA_sel                    <= "00";
388     dq_sel                    <= "00";
389     dq_OE                     <= '0';
390     writedata_load            <= '0';
391     addresssgen_enable        <= '0';
392     synch_enable              <= '0';
393     synch_clear_n             <= '1';
394     RWDS_sampling_enable      <= '0';
395     check_latency             <= '0';
396     force_RWDS_low            <= '0';
397     hCK_gating_enable_n       <= '1';
398     set_dpd_status            <= '0';
399     clear_dpd_status_n        <= '1';
400     deadline_tim_enable       <= '0';
401     deadline_tim_clear_n      <= '1';
402     hCKen_pipe_clear_n        <= '1';
403     hbus_RESET_n              <= '1';
404     hbus_CS_n                 <= '1';
405 -----
406 case present_state is
407 -----
408     when reset =>
409         waitrequest                <= '1';
410         hCK_gating_enable_n         <= '0';
411         hbus_RESET_n                <= '0';
412         deadline_tim_clear_n        <= '0';
413         clear_dpd_status_n          <= '0';
414         synch_clear_n               <= '0';
415         writedata_load              <= '1';
416         dpd_req_clear_n             <= '0';
417         reset_config_register_n     <= '0';
418         hCKen_pipe_clear_n          <= '0';
419 -----
420     when reset_wait =>
421         waitrequest                <= '1';
422         hCK_gating_enable_n         <= '0';
423         deadline_tim_enable         <= '1';
424         hbus_RESET_n                <= '0';
425 -----
426     when reset_exit_begin =>
427         waitrequest                <= '1';
428         hCK_gating_enable_n         <= '0';
429         deadline_tim_clear_n        <= '0';
430 -----
431     when reset_exit =>
432         waitrequest                <= '1';

```

```

433     hCK_gating_enable_n      <= '0';
434     deadline_tim_enable     <= '1';
435 -----
436 when idle =>
437     hCK_gating_enable_n      <= '0';
438     deadline_tim_clear_n     <= '0';
439     cmd_load                 <= '1';
440     datain_load             <= '1';
441 -----
442 when dummycmd =>
443     waitrequest              <= '1';
444     hbus_CS_n               <= '0';
445     deadline_tim_enable     <= '1';
446 -----
447 when dummycmd_last | dummycmd_end =>
448     waitrequest              <= '1';
449     hbus_CS_n               <= '0';
450     hCK_gating_enable_n     <= '0';
451     deadline_tim_clear_n    <= '0';
452 -----
453 when wait_dpd_out =>
454     waitrequest              <= '1';
455     hCK_gating_enable_n     <= '0';
456     deadline_tim_enable     <= '1';
457 -----
458 when write_virtconf =>
459     waitrequest              <= '1';
460     hCK_gating_enable_n     <= '0';
461     clear_dpd_status_n      <= '0';
462     update_config_register  <= '1';
463 -----
464 when writeconf0_prep =>
465     waitrequest              <= '1';
466     hbus_CS_n               <= '0';
467     CA_load                 <= '1';
468     config_access           <= '1';
469     address_space_sel       <= "01";
470 -----
471 when writeconf_CA0 =>
472     waitrequest              <= '1';
473     hbus_CS_n               <= '0';
474     dq_sel                  <= "11";
475     writedata_load          <= '1';
476     CA_sel                  <= "00";
477 -----
478 when writeconf_CA1 =>
479     waitrequest              <= '1';
480     hbus_CS_n               <= '0';
481     dq_OE                   <= '1';
482     dq_sel                  <= "11";
483     writedata_load          <= '1';
484     CA_sel                  <= "01";
485 -----
486 when writeconf_CA2 =>
487     waitrequest              <= '1';
488     hbus_CS_n               <= '0';
489     dq_OE                   <= '1';
490     dq_sel                  <= "11";
491     writedata_load          <= '1';
492     CA_sel                  <= "10";
493 -----
494 when writeconf0 =>
495     waitrequest              <= '1';
496     hCK_gating_enable_n     <= '0';
497     hbus_CS_n               <= '0';
498     writedata_load          <= '1';
499     dq_OE                   <= '1';
500     dq_sel                  <= "01";
501 -----
502 when writeconf0_end =>
503     waitrequest              <= '1';
504     hbus_CS_n               <= '0';
505     dq_OE                   <= '1';
506     hCK_gating_enable_n     <= '0';
507 -----
508 when wait_dpd_in =>

```

```

509     waitrequest                <= '1';
510     hCK_gating_enable_n       <= '0';
511     deadline_tim_enable       <= '1';
512     set_dpd_status            <= '1';
513
514     -----
514     when read_virtconf =>
515         hCK_gating_enable_n     <= '0';
516         avs_out_sel             <= '1';
517         force_valid             <= '1';
518         cmd_load                <= '1';
519         datain_load             <= '1';
520
521     -----
521     when readmem_prep =>
522         waitrequest              <= '1';
523         hbus_CS_n                <= '0';
524         CA_load                  <= '1';
525         address_space_sel        <= "00";
526         read_writeN             <= '1';
527
528     -----
528     when writemem_prep =>
529         waitrequest              <= '1';
530         hbus_CS_n                <= '0';
531         CA_load                  <= '1';
532         address_space_sel        <= "00";
533         synch_cnt_enable         <= '1';
534
535     -----
535     when CA_0 =>
536         waitrequest              <= '1';
537         hbus_CS_n                <= '0';
538         dq_sel                   <= "11";
539         writedata_load           <= '1';
540         CA_sel                   <= "00";
541         synch_cnt_enable         <= '1';
542
543     -----
543     when CA_1 =>
544         waitrequest              <= '1';
545         hbus_CS_n                <= '0';
546         dq_OE                    <= '1';
547         dq_sel                   <= "11";
548         writedata_load           <= '1';
549         CA_sel                   <= "01";
550
551     -----
551     when CA_2 =>
552         waitrequest              <= '1';
553         hbus_CS_n                <= '0';
554         dq_OE                    <= '1';
555         dq_sel                   <= "11";
556         writedata_load           <= '1';
557         CA_sel                   <= "10";
558         deadline_tim_enable      <= '1';
559         check_latency            <= '1';
560
561     -----
561     when CA_end =>
562         waitrequest              <= '1';
563         hbus_CS_n                <= '0';
564         dq_OE                    <= '1';
565         deadline_tim_enable      <= '1';
566
567     -----
567     when read_wait_0 =>
568         waitrequest              <= '1';
569         hbus_CS_n                <= '0';
570         RWDS_sampling_enable     <= '1';
571         synch_enable             <= '1';
572         synch_cnt_clear_n        <= '0';
573
574     -----
574     when read_wait_1 | read_wait_2 =>
575         waitrequest              <= '1';
576         hbus_CS_n                <= '0';
577         RWDS_sampling_enable     <= '1';
578         synch_enable             <= '1';
579
580     -----
580     when read_end_1 | read_end_2 =>
581         waitrequest              <= '1';
582         hbus_CS_n                <= '0';
583         hCK_gating_enable_n      <= '0';
584
585     -----

```



```

585     when synch_restoring_1 =>
586         waitrequest           <= '1';
587         hCK_gating_enable_n   <= '0';
588         synch_clear_n         <= '0';
589     -----
590     when synch_restoring_2 =>
591         waitrequest           <= '1';
592         hCK_gating_enable_n   <= '0';
593     -----
594     when writemem_wait =>
595         waitrequest           <= '1';
596         hbus_CS_n             <= '0';
597         deadline_tim_enable   <= '1';
598     -----
599     when writemem =>
600         waitrequest           <= '1';
601         hCK_gating_enable_n   <= '0';
602         hbus_CS_n             <= '0';
603         writedata_load        <= '1';
604         dq_sel                 <= "00";
605     -----
606     when write_end =>
607         waitrequest           <= '1';
608         hbus_CS_n             <= '0';
609         hCK_gating_enable_n   <= '0';
610         force_RWDS_low        <= '1';
611         synch_cnt_clear_n     <= '0';
612         deadline_tim_clear_n  <= '0';
613         dq_OE                  <= '1';
614     -----
615     when writeburst_prep =>
616         hbus_CS_n             <= '0';
617         writedata_load        <= '1';
618         dq_sel                 <= "00";
619         datain_load           <= '1';
620         synch_cnt_enable      <= '1';
621     -----
622     when writeburst =>
623         hbus_CS_n             <= '0';
624         writedata_load        <= '1';
625         dq_sel                 <= "00";
626         dq_OE                  <= '1';
627         datain_load           <= '1';
628         synch_cnt_enable      <= '1';
629         force_RWDS_low        <= '1';
630     -----
631     when writeburst_last =>
632         waitrequest           <= '1';
633         hbus_CS_n             <= '0';
634         hCK_gating_enable_n   <= '0';
635         writedata_load        <= '1';
636         dq_sel                 <= "00";
637         dq_OE                  <= '1';
638         force_RWDS_low        <= '1';
639     -----
640     when stop_burst_1 =>
641         waitrequest           <= '1';
642         hbus_CS_n             <= '0';
643         hCK_gating_enable_n   <= '0';
644         synch_cnt_enable      <= '1';
645         synch_cnt_down        <= '1';
646         addressgen_enable     <= '1';
647         force_RWDS_low        <= '1';
648     -----
649     when stop_burst_2 =>
650         waitrequest           <= '1';
651         hbus_CS_n             <= '0';
652         hCK_gating_enable_n   <= '0';
653     -----
654     when idle_burst =>
655         hCK_gating_enable_n   <= '0';
656         deadline_tim_clear_n  <= '0';
657         datain_load           <= '1';
658     -----
659     when restore_burst =>
660         waitrequest           <= '1';

```

```

661         hbus_CS_n           <= '0';
662         CA_load             <= '1';
663         address_space_sel   <= "11";
664         -----
665     end case;
666 end process control_signals_definition; -----
667
668 end architecture fsm;

```

```

1  -- BRIEF DESCRIPTION: Avalon memory-mapped slave to hyperRAM converter
2
3  library     IEEE;
4  use        IEEE.std_logic_1164.all;
5  use        IEEE.numeric_std.all;
6
7  entity avs_hram_mainconv is
8  generic
9  (
10     drive_strength : std_logic_vector(2 downto 0) := "000"
11 );
12 port
13 (
14     clk           : in    std_logic;
15     reset_n       : in    std_logic;
16     -- avs signals
17     avs_address   : in    std_logic_vector(22 downto 0);
18     avs_read      : in    std_logic;
19     avs_readdata  : out   std_logic_vector(15 downto 0);
20     avs_write     : in    std_logic;
21     avs_writedata : in    std_logic_vector(15 downto 0);
22     avs_waitrequest : out  std_logic;
23     avs_readdatavalid : out std_logic;
24     avs_burstcount : in   std_logic_vector(10 downto 0);
25     -- hram signals
26     hram_RESET_n  : out   std_logic;
27     hram_DQ       : inout  std_logic_vector(7 downto 0);
28     hram_RWDS     : inout  std_logic;
29     hram_CS_n     : out    std_logic;
30     clk90         : out    std_logic;
31     hCK_enable    : out    std_logic
32 );
33 end entity avs_hram_mainconv;
34
35 architecture rtl of avs_hram_mainconv is
36
37     -- COMPONENT: execution unit -----
38     component avs_hram_mainconv_EU is
39     generic
40     (
41         drive_strength : std_logic_vector(2 downto 0) := "000"
42     );
43     port
44     (
45         clk           : in    std_logic;
46         reset_n       : in    std_logic;
47         -- avs signals
48         avs_address   : in    std_logic_vector(22 downto 0);
49         avs_read      : in    std_logic;
50         avs_readdata  : out   std_logic_vector(15 downto 0);
51         avs_write     : in    std_logic;
52         avs_writedata : in    std_logic_vector(15 downto 0);
53         avs_waitrequest : out  std_logic;
54         avs_readdatavalid : out std_logic;
55         avs_burstcount : in   std_logic_vector(10 downto 0);
56         -- hram signals
57         hram_RESET_n  : out   std_logic;
58         hram_DQ       : inout  std_logic_vector(7 downto 0);
59         hram_RWDS     : inout  std_logic;
60         hram_CS_n     : out    std_logic;
61         clk90         : out    std_logic;
62         hCK_enable    : out    std_logic;
63         -- control signals
64         waitrequest   : in    std_logic;
65         force_valid   : in    std_logic;

```

```

66     cmd_load           : in   std_logic;
67     dpd_req_clear_n   : in   std_logic;
68     synch_cnt_enable  : in   std_logic;
69     synch_cnt_down    : in   std_logic;
70     synch_cnt_clear_n : in   std_logic;
71     datain_load       : in   std_logic;
72     avs_out_sel       : in   std_logic;
73     reset_config_register_n : in std_logic;
74     update_config_register : in std_logic;
75     address_space_sel  : in   std_logic_vector(1 downto 0);
76     config_access      : in   std_logic;
77     read_writeN       : in   std_logic;
78     CA_load           : in   std_logic;
79     CA_sel            : in   std_logic_vector(1 downto 0);
80     dq_sel            : in   std_logic_vector(1 downto 0);
81     dq_OE             : in   std_logic;
82     writedata_load    : in   std_logic;
83     addressgen_enable  : in   std_logic;
84     synch_enable       : in   std_logic;
85     synch_clear_n     : in   std_logic;
86     RWDS_sampling_enable : in std_logic;
87     check_latency     : in   std_logic;
88     force_RWDS_low    : in   std_logic;
89     hCK_gating_enable_n : in std_logic;
90     set_dpd_status    : in   std_logic;
91     clear_dpd_status_n : in   std_logic;
92     deadline_tim_enable : in   std_logic;
93     deadline_tim_clear_n : in std_logic;
94     hCKen_pipe_clear_n : in   std_logic;
95     hbus_RESET_n     : in   std_logic;
96     hbus_CS_n        : in   std_logic;
97     -- status signals
98     write              : out  std_logic;
99     read               : out  std_logic;
100    config             : out  std_logic;
101    dpd_req            : out  std_logic;
102    active_dpd_req     : out  std_logic;
103    current_operation  : out  std_logic;
104    bursttransfer      : out  std_logic;
105    burst_end          : out  std_logic;
106    synch_busy         : out  std_logic;
107    doubled_latency    : out  std_logic;
108    dpd_mode_on        : out  std_logic;
109    t_acc1             : out  std_logic;
110    t_acc2             : out  std_logic;
111    t_dpdcsl          : out  std_logic;
112    t_dpdin           : out  std_logic;
113    t_dpdpout         : out  std_logic;
114 );
115 end component; -----
116
117 -- COMPONENT: control unit -----
118 component avs_hram_mainconv_CU is
119 port
120 (
121     clk                : in   std_logic;
122     reset_n            : in   std_logic;
123     -- control signals
124     waitrequest        : out  std_logic;
125     force_valid        : out  std_logic;
126     cmd_load           : out  std_logic;
127     dpd_req_clear_n   : out  std_logic;
128     synch_cnt_enable  : out  std_logic;
129     synch_cnt_down    : out  std_logic;
130     synch_cnt_clear_n : out  std_logic;
131     datain_load       : out  std_logic;
132     avs_out_sel       : out  std_logic;
133     reset_config_register_n : out std_logic;
134     update_config_register : out std_logic;
135     address_space_sel  : out  std_logic_vector(1 downto 0);
136     config_access      : out  std_logic;
137     read_writeN       : out  std_logic;
138     CA_load           : out  std_logic;
139     CA_sel            : out  std_logic_vector(1 downto 0);
140     dq_sel            : out  std_logic_vector(1 downto 0);
141     dq_OE             : out  std_logic;

```

```

142 writedata_load      : out std_logic;
143 addressgen_enable   : out std_logic;
144 synch_enable        : out std_logic;
145 synch_clear_n       : out std_logic;
146 RWDS_sampling_enable : out std_logic;
147 check_latency       : out std_logic;
148 force_RWDS_low      : out std_logic;
149 hCK_gating_enable_n : out std_logic;
150 set_dpd_status       : out std_logic;
151 clear_dpd_status_n   : out std_logic;
152 deadline_tim_enable  : out std_logic;
153 deadline_tim_clear_n : out std_logic;
154 hCKen_pipe_clear_n  : out std_logic;
155 hbus_RESET_n        : out std_logic;
156 hbus_CS_n           : out std_logic;
157 -- status signals
158 write                : in  std_logic;
159 read                 : in  std_logic;
160 config               : in  std_logic;
161 dpd_req              : in  std_logic;
162 active_dpd_req       : in  std_logic;
163 current_operation    : in  std_logic;
164 burstttransfer       : in  std_logic;
165 burst_end            : in  std_logic;
166 synch_busy           : in  std_logic;
167 doubled_latency     : in  std_logic;
168 dpd_mode_on         : in  std_logic;
169 t_acc1               : in  std_logic;
170 t_acc2               : in  std_logic;
171 t_dpdcsl             : in  std_logic;
172 t_dpdin              : in  std_logic;
173 t_dpdpout            : in  std_logic;
174 );
175 end component;-----
176
177 -- SIGNALS -----
178 signal waitrequest      : std_logic;
179 signal force_valid      : std_logic;
180 signal cmd_load         : std_logic;
181 signal dpd_req_clear_n  : std_logic;
182 signal synch_cnt_enable : std_logic;
183 signal synch_cnt_down   : std_logic;
184 signal synch_cnt_clear_n : std_logic;
185 signal datain_load      : std_logic;
186 signal avs_out_sel      : std_logic;
187 signal reset_config_register_n : std_logic;
188 signal update_config_register : std_logic;
189 signal address_space_sel : std_logic_vector(1 downto 0);
190 signal config_access    : std_logic;
191 signal read_writeN      : std_logic;
192 signal CA_load          : std_logic;
193 signal CA_sel           : std_logic_vector(1 downto 0);
194 signal dq_sel           : std_logic_vector(1 downto 0);
195 signal dq_OE            : std_logic;
196 signal writedata_load   : std_logic;
197 signal addressgen_enable : std_logic;
198 signal synch_enable     : std_logic;
199 signal synch_clear_n    : std_logic;
200 signal RWDS_sampling_enable : std_logic;
201 signal check_latency    : std_logic;
202 signal force_RWDS_low   : std_logic;
203 signal hCK_gating_enable_n : std_logic;
204 signal set_dpd_status   : std_logic;
205 signal clear_dpd_status_n : std_logic;
206 signal deadline_tim_enable : std_logic;
207 signal deadline_tim_clear_n : std_logic;
208 signal hCKen_pipe_clear_n : std_logic;
209 signal hbus_RESET_n    : std_logic;
210 signal hbus_CS_n       : std_logic;
211 signal write            : std_logic;
212 signal read             : std_logic;
213 signal config           : std_logic;
214 signal dpd_req          : std_logic;
215 signal active_dpd_req   : std_logic;
216 signal current_operation : std_logic;
217 signal burstttransfer   : std_logic;

```

```

218 signal burst_end           : std_logic;
219 signal synch_busy          : std_logic;
220 signal doubled_latency    : std_logic;
221 signal dpd_mode_on        : std_logic;
222 signal t_acc1              : std_logic;
223 signal t_acc2              : std_logic;
224 signal t_dpdcsl           : std_logic;
225 signal t_dpdin             : std_logic;
226 signal t_dpdpout          : std_logic;
227 -----
228
229 begin
230
231 -- EU -----
232 EU : avs_hram_mainconv_EU
233 generic map
234 (
235     drive_strength => drive_strength
236 )
237 port map
238 (
239     clk                => clk,
240     reset_n            => reset_n,
241     avs_address        => avs_address,
242     avs_read           => avs_read,
243     avs_readdata       => avs_readdata,
244     avs_write          => avs_write,
245     avs_writedata      => avs_writedata,
246     avs_waitrequest    => avs_waitrequest,
247     avs_readdatavalid => avs_readdatavalid,
248     avs_burstcount     => avs_burstcount,
249     hram_RESET_n      => hram_RESET_n,
250     hram_DQ            => hram_DQ,
251     hram_RWDS          => hram_RWDS,
252     hram_CS_n          => hram_CS_n,
253     clk90              => clk90,
254     hCK_enable         => hCK_enable,
255     waitrequest        => waitrequest,
256     force_valid        => force_valid,
257     cmd_load           => cmd_load,
258     dpd_req_clear_n   => dpd_req_clear_n,
259     synch_cnt_enable   => synch_cnt_enable,
260     synch_cnt_down     => synch_cnt_down,
261     synch_cnt_clear_n => synch_cnt_clear_n,
262     datain_load        => datain_load,
263     avs_out_sel        => avs_out_sel,
264     reset_config_register_n => reset_config_register_n,
265     update_config_register => update_config_register,
266     address_space_sel  => address_space_sel,
267     config_access      => config_access,
268     read_writeN        => read_writeN,
269     CA_load            => CA_load,
270     CA_sel             => CA_sel,
271     dq_sel             => dq_sel,
272     dq_OE              => dq_OE,
273     writedata_load     => writedata_load,
274     addressgen_enable  => addressgen_enable,
275     synch_enable       => synch_enable,
276     synch_clear_n      => synch_clear_n,
277     RWDS_sampling_enable => RWDS_sampling_enable,
278     check_latency      => check_latency,
279     force_RWDS_low     => force_RWDS_low,
280     hCK_gating_enable_n => hCK_gating_enable_n,
281     set_dpd_status     => set_dpd_status,
282     clear_dpd_status_n => clear_dpd_status_n,
283     deadline_tim_enable => deadline_tim_enable,
284     deadline_tim_clear_n => deadline_tim_clear_n,
285     hCKen_pipe_clear_n => hCKen_pipe_clear_n,
286     hbus_RESET_n       => hbus_RESET_n,
287     hbus_CS_n          => hbus_CS_n,
288     write              => write,
289     read               => read,
290     config             => config,
291     dpd_req            => dpd_req,
292     active_dpd_req     => active_dpd_req,
293     current_operation  => current_operation,

```

```

294     bursttransfer      => bursttransfer ,
295     burst_end         => burst_end ,
296     synch_busy        => synch_busy ,
297     doubled_latency   => doubled_latency ,
298     dpd_mode_on      => dpd_mode_on ,
299     t_acc1            => t_acc1 ,
300     t_acc2            => t_acc2 ,
301     t_dpdcsl         => t_dpdcsl ,
302     t_dpdin           => t_dpdin ,
303     t_dp dout         => t_dp dout
304 ); -----
305
306 -- CU -----
307 CU : avs_hram_mainconv_CU
308 port map
309 (
310     clk                => clk ,
311     reset_n           => reset_n ,
312     waitrequest       => waitrequest ,
313     force_valid       => force_valid ,
314     cmd_load          => cmd_load ,
315     dpd_req_clear_n   => dpd_req_clear_n ,
316     synch_cnt_enable  => synch_cnt_enable ,
317     synch_cnt_down    => synch_cnt_down ,
318     synch_cnt_clear_n => synch_cnt_clear_n ,
319     datain_load       => datain_load ,
320     avs_out_sel        => avs_out_sel ,
321     reset_config_register_n => reset_config_register_n ,
322     update_config_register => update_config_register ,
323     address_space_sel => address_space_sel ,
324     config_access     => config_access ,
325     read_writeN       => read_writeN ,
326     CA_load           => CA_load ,
327     CA_sel            => CA_sel ,
328     dq_sel            => dq_sel ,
329     dq_OE             => dq_OE ,
330     writedata_load    => writedata_load ,
331     addressgen_enable => addressgen_enable ,
332     synch_enable      => synch_enable ,
333     synch_clear_n     => synch_clear_n ,
334     RWDS_sampling_enable => RWDS_sampling_enable ,
335     check_latency     => check_latency ,
336     force_RWDS_low   => force_RWDS_low ,
337     hCK_gating_enable_n => hCK_gating_enable_n ,
338     set_dpd_status    => set_dpd_status ,
339     clear_dpd_status_n => clear_dpd_status_n ,
340     deadline_tim_enable => deadline_tim_enable ,
341     deadline_tim_clear_n => deadline_tim_clear_n ,
342     hCKen_pipe_clear_n => hCKen_pipe_clear_n ,
343     hbus_RESET_n     => hbus_RESET_n ,
344     hbus_CS_n        => hbus_CS_n ,
345     write             => write ,
346     read              => read ,
347     config            => config ,
348     dpd_req           => dpd_req ,
349     active_dpd_req    => active_dpd_req ,
350     current_operation => current_operation ,
351     bursttransfer     => bursttransfer ,
352     burst_end         => burst_end ,
353     synch_busy        => synch_busy ,
354     doubled_latency   => doubled_latency ,
355     dpd_mode_on      => dpd_mode_on ,
356     t_acc1            => t_acc1 ,
357     t_acc2            => t_acc2 ,
358     t_dpdcsl         => t_dpdcsl ,
359     t_dpdin           => t_dpdin ,
360     t_dp dout         => t_dp dout
361 ); -----
362
363 end architecture rtl;

```

BIBLIOGRAPHY

- [1] Massimo Ruo Roch, Maurizio Martina, *VirtLAB: a Low-Cost Platform for Electronic Lab experiments*, Sensors
- [2] Cypress Semiconductor Corporation, *S27KL0641/S27KS0641/S70KL1281/S70KS1281, 3.0V/1.8V, 64Mb(8MB)/128Mb(16MB), HyperRAM™ Self-Refresh DRAM*, 001-97964 Rev. *M
- [3] Intel, *Avalon® Interface Specifications*, 683091, 2022.01.24
- [4] Intel, *Intel® Cyclone® 10 LP Device Datasheet*, 683251, 2022.10.31