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

CONTENTS

ACRONYMS

DDR Double Data Rate

FPGA Field Programmable Gate Array

IP Intellectual Property

MCU MicroController Unit

RAM Random Access Memory

SDR Single Data Rate

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 is already programmed in order to provide a virtual replacement of the bench equipment.

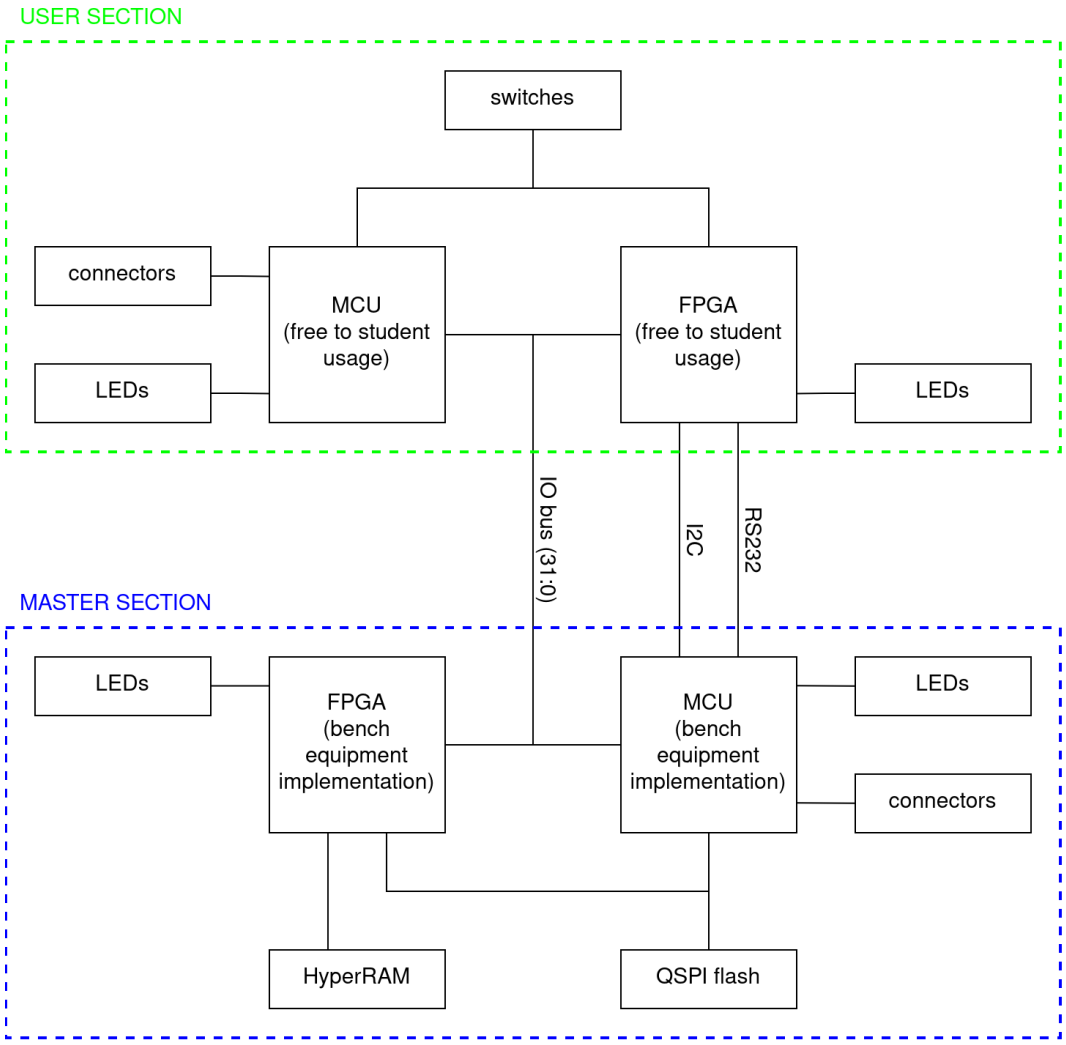

Figure 1.1: VirtLAB board block diagram

Currently, it is still not possible to exploit the master section to its full potential. In particular, the FPGA can still not be properly used to implement many of the useful applications it may realize. 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, we can 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, developement 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.

2.1 Avalon Memory Mapped Interface

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

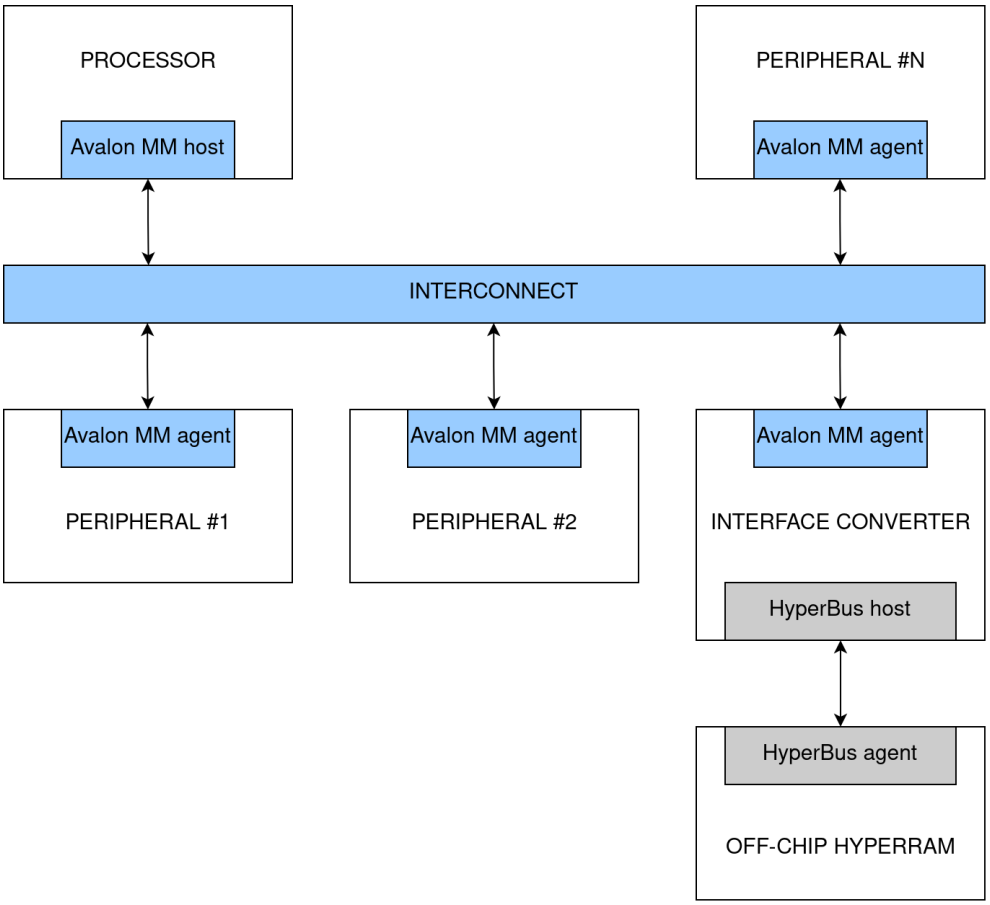

Figure 2.1: Typical Host-Agent system using components with an Avalon Memory Mapped interface (highlighted in light blue). The HyperRAM can be connected to the system only by using a suitable interface converter.

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, we have to make the following considerations:

- 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.

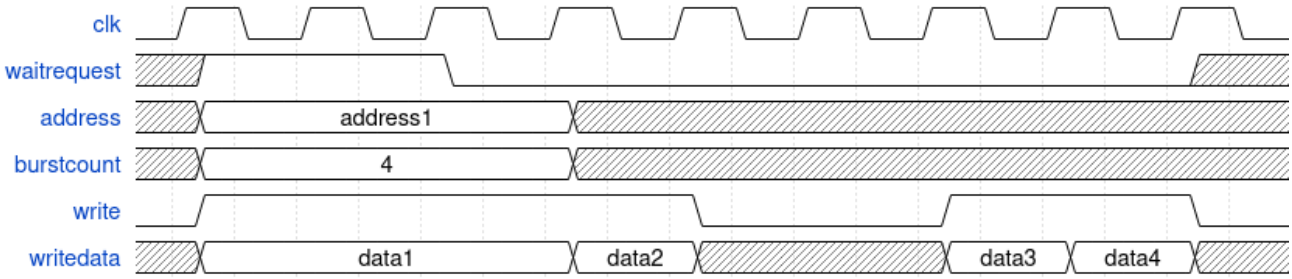

Figure 2.2: Avalon Memory Mapped interface - write operation timing diagram

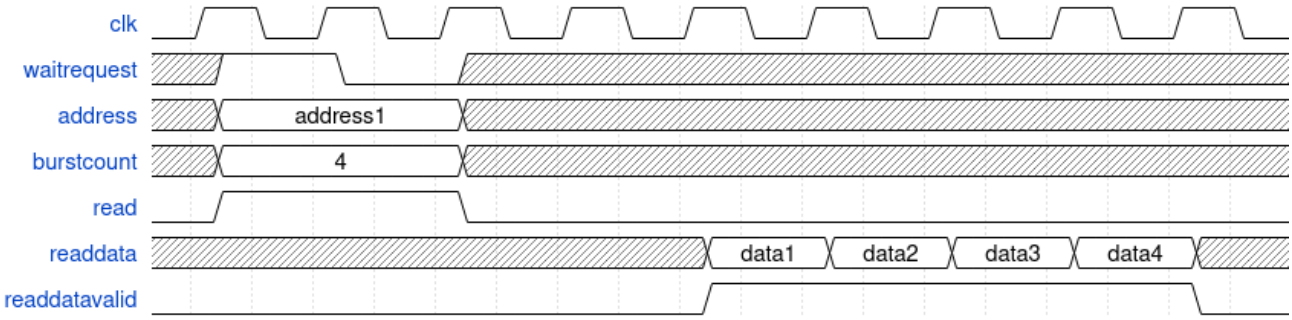

Figure 2.3: Avalon Memory Mapped interface, read operation timing diagram

2.2 HyperRAM Interface

The HyperRAM interface is based on an 8-bit DDR data bus used to transfer data, addresses and commands. It 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.

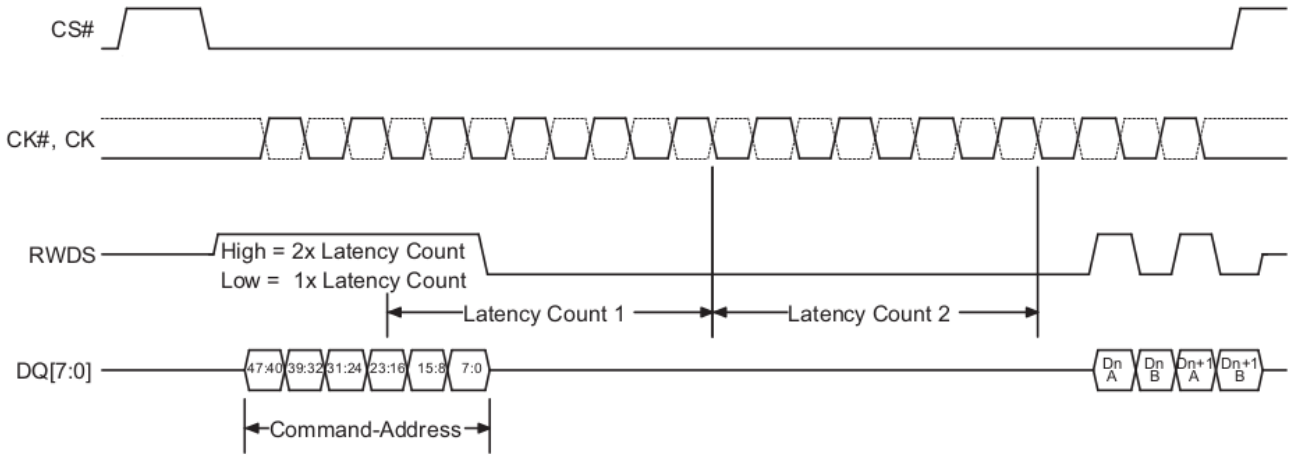

Figure 2.4: HyperRAM interface, read operation timing diagram. During the command transfer, the host drives DQ and the memory drives RWDS. During the data transfer, the memory drives both DQ and RWDS.

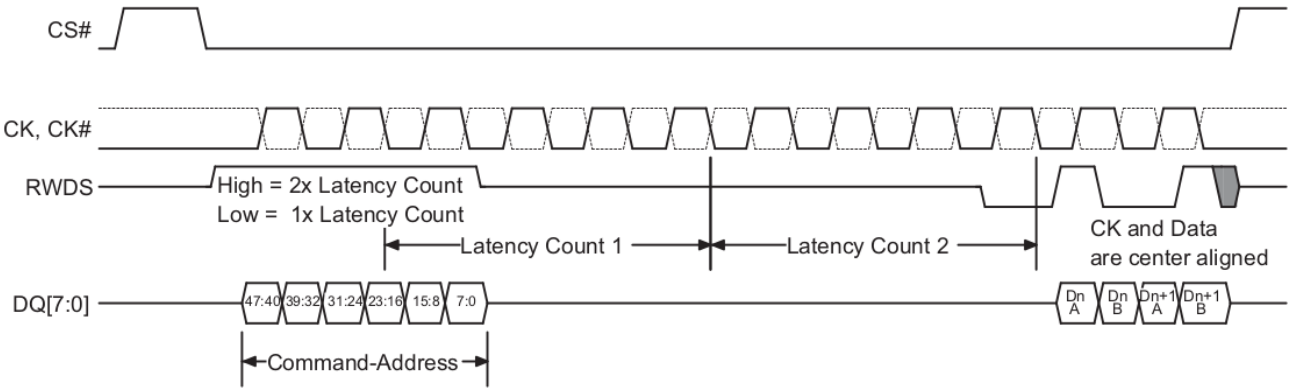

Figure 2.5: HyperRAM interface, write operation timing diagram. During the command transfer, the host drives DQ and the memory drives RWDS. During the data transfer, the host drives both DQ and RWDS.

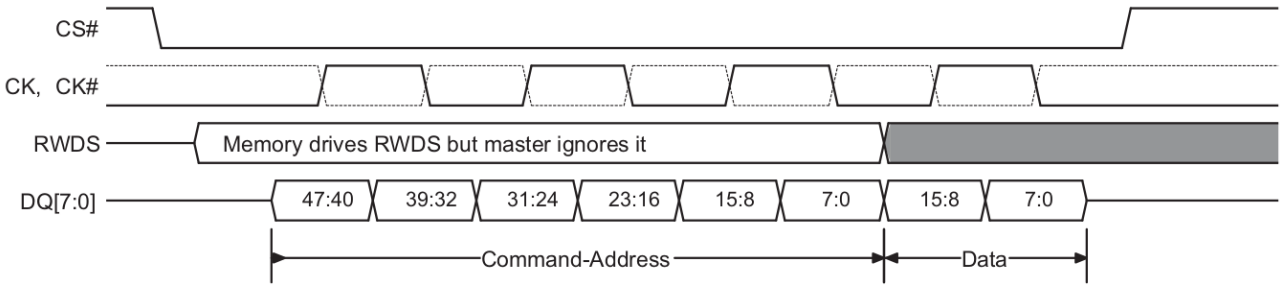

Figure 2.6: HyperRAM interface, register write operation timing diagram. DQ is always driven by the host.

2.3 HyperRAM Specifications

2.3.1 Command-Address

As we saw in section 2.2, the operation command and the addres 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 2.7: Command-Address (CA) bit assignment

2.3.2 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 2.8: 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 µs for Industrial temperature range devices 1 µs for Industrial Plus temperature range devices 11b — 1.5 times default 00b — 2 times default 01b — 4 times default

Figure 2.9: Configuration Register 1 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 2.10: Configuration Register 0 bit assignment

2.3.3 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.

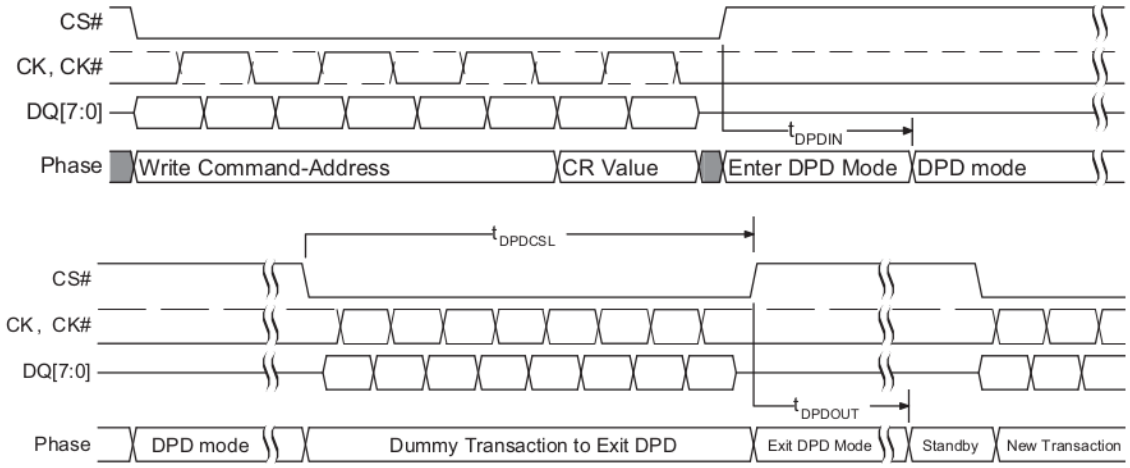

Figure 2.11: DPD timing diagram

2.3.4 Power-Up

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

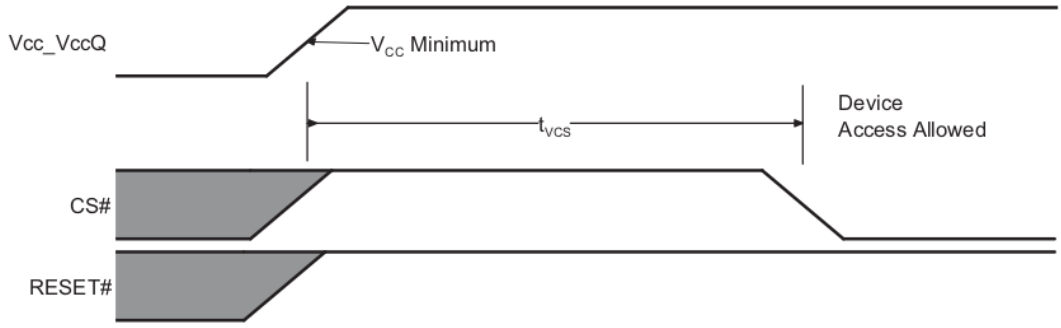

Figure 2.12: Power-up with *RESET#* high

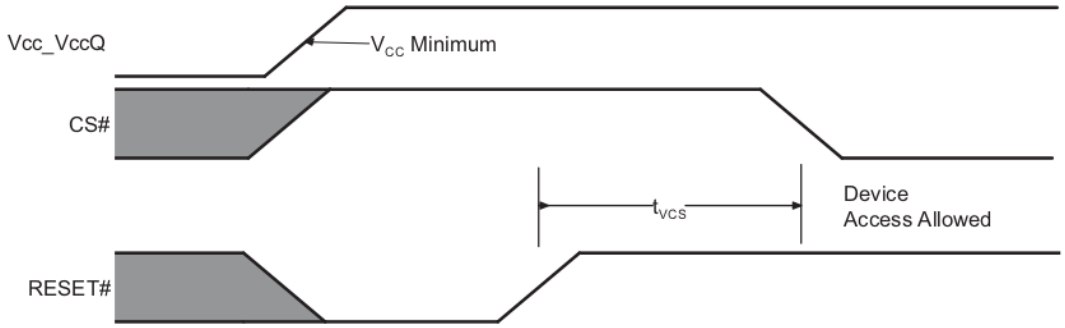

Figure 2.13: Power-up with *RESET#* low

2.3.5 Timing Specifications

During a read/write operation, several timing parameters shall be respected:

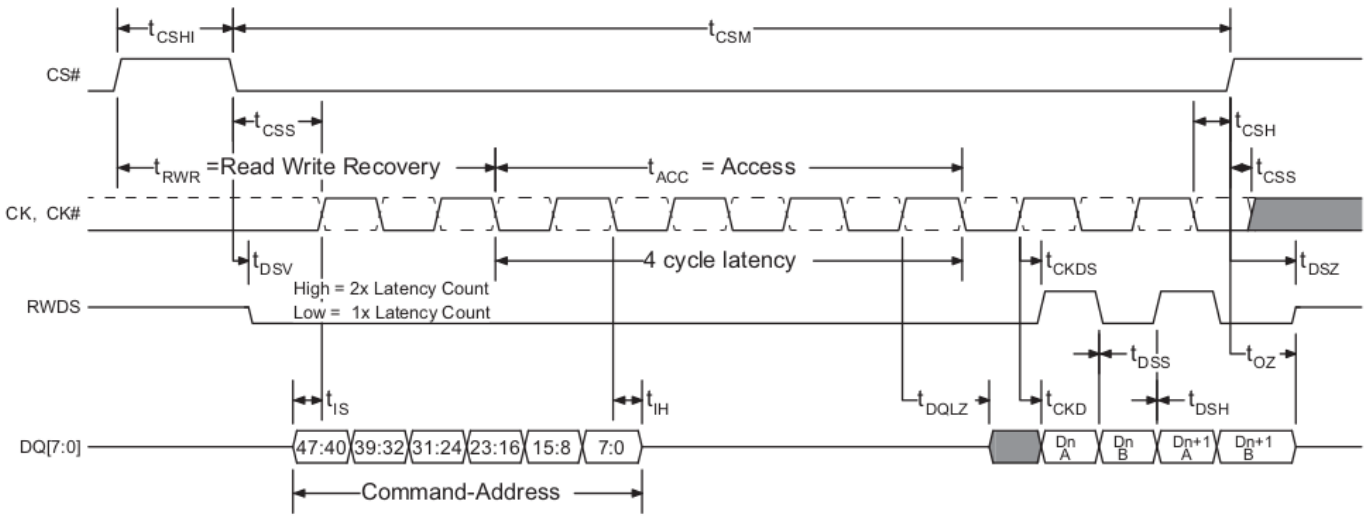

Figure 2.14: Read timing parameters

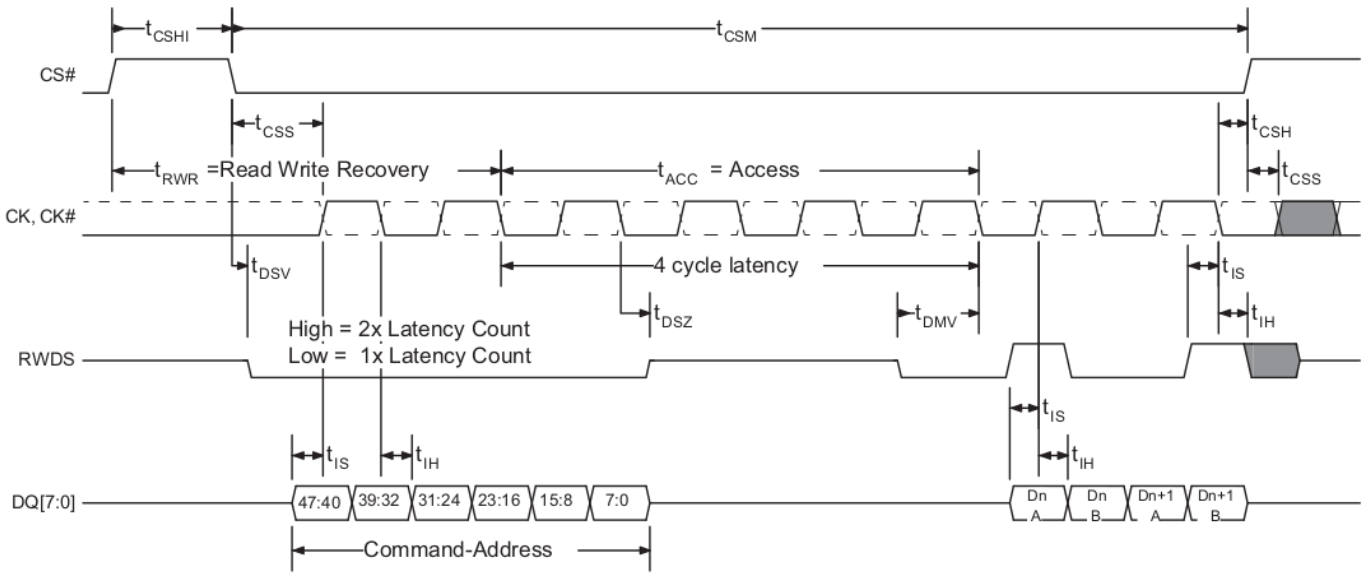

Figure 2.15: Write timing parameters

2.4 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, register space access requires dedicated addresses that are not in conflict with the ones related to the memory location.

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 using the following approach:

- 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 (default) 1: enter DPD mode
1	Fixed Latency Enable	0: variable latency (default) 1: fixed latency
2-15	Reserved	Reserved for future expansions.

As far as the frequency is concerned, the S27KL0641DA 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 (chapter ?? section ??).

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) and the HDL model of the HyperRAM provided by the manufacturer to create an Avalon Memory Mapped system suitable for the test. Overall, the test environment is an processor-based system that read some inputs and change the status of some LEDs according to it. The HyperRAM is employed as data memory for the processor.

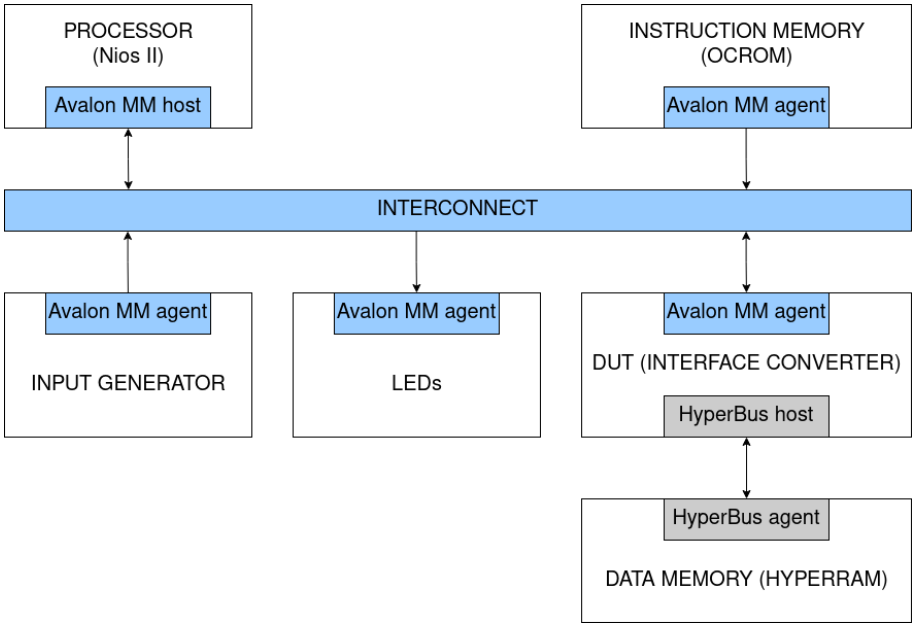

Figure 3.1: Test environment for the interface converter

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, that can be obtained simply by using an IP core provided by the CAD software.

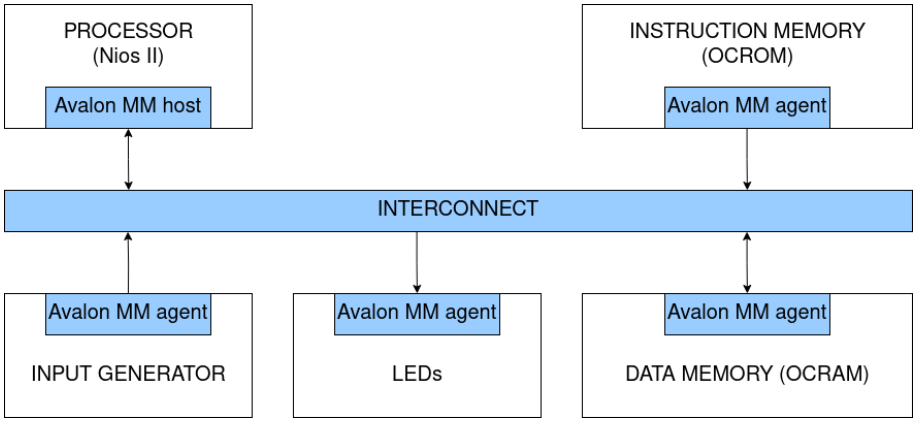

Figure 3.2: Test environment verification

DESIGN PARTITIONING

Apart from clock and reset, the interface converter deals just with the Avalon signals and the HyperRAM signals. 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 2.4. The burstcount signal is on 11 bits, i.e the maximum possible parallelism, corresponding to a theoretical maximum burst lenght equal to 2^{10} as stated in the Avalon documentation.

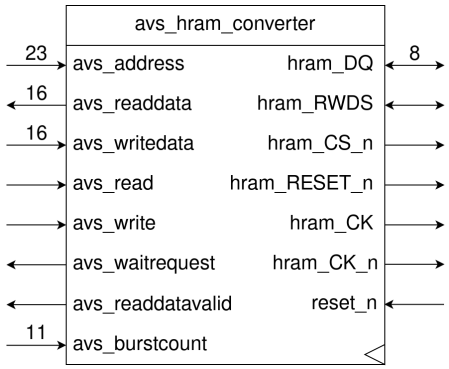

Figure 4.1: Blackbox of the interface converter

Unfortunately, it is usually not possible to push the burst lenght up to the theoretical maximum, since the duration of any memory operation is upper bounded. Considering that the latency of the converter depends on its implementation and that the memory access time can vary, it is not possible to estimate the actual upper bound of the burst lenght in advance. For this reason, the maximum parallelism is employed and the actual maximum lenght will be estimated in different conditions by means of simulations (chapter 5).

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 in order to put it on the memory data bus.
- **DDR to SDR conversion and synchronization**: 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 internal clock.
- **Clock shifting and clock gating**: the internal clock must by 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 already provides an IP implementing a clock controller. Indeed, we just have to create a custom IP (*avs_hram_mainconv* in figure 4.2) implementing all the features but the clock gating and combine it with the clock controller IP (*clkctrl* in figure 4.2) to create the interface controller (*avs_hram_converter* in figure 4.1). The architecture of the *avs_hram_converter* IP is represented in figure 4.3:

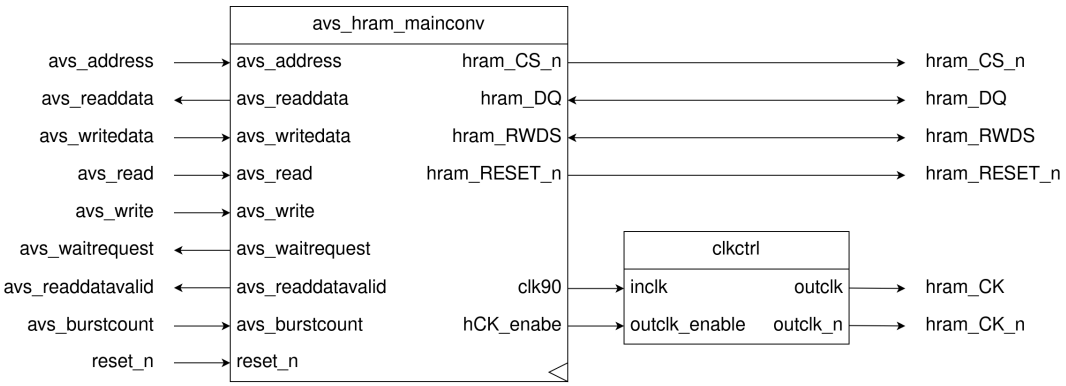

Figure 4.2: Combination of the clock controller IP implementing the clock gating (already provided by the CAD software) and the custom IP implementing all the other required functionalities.

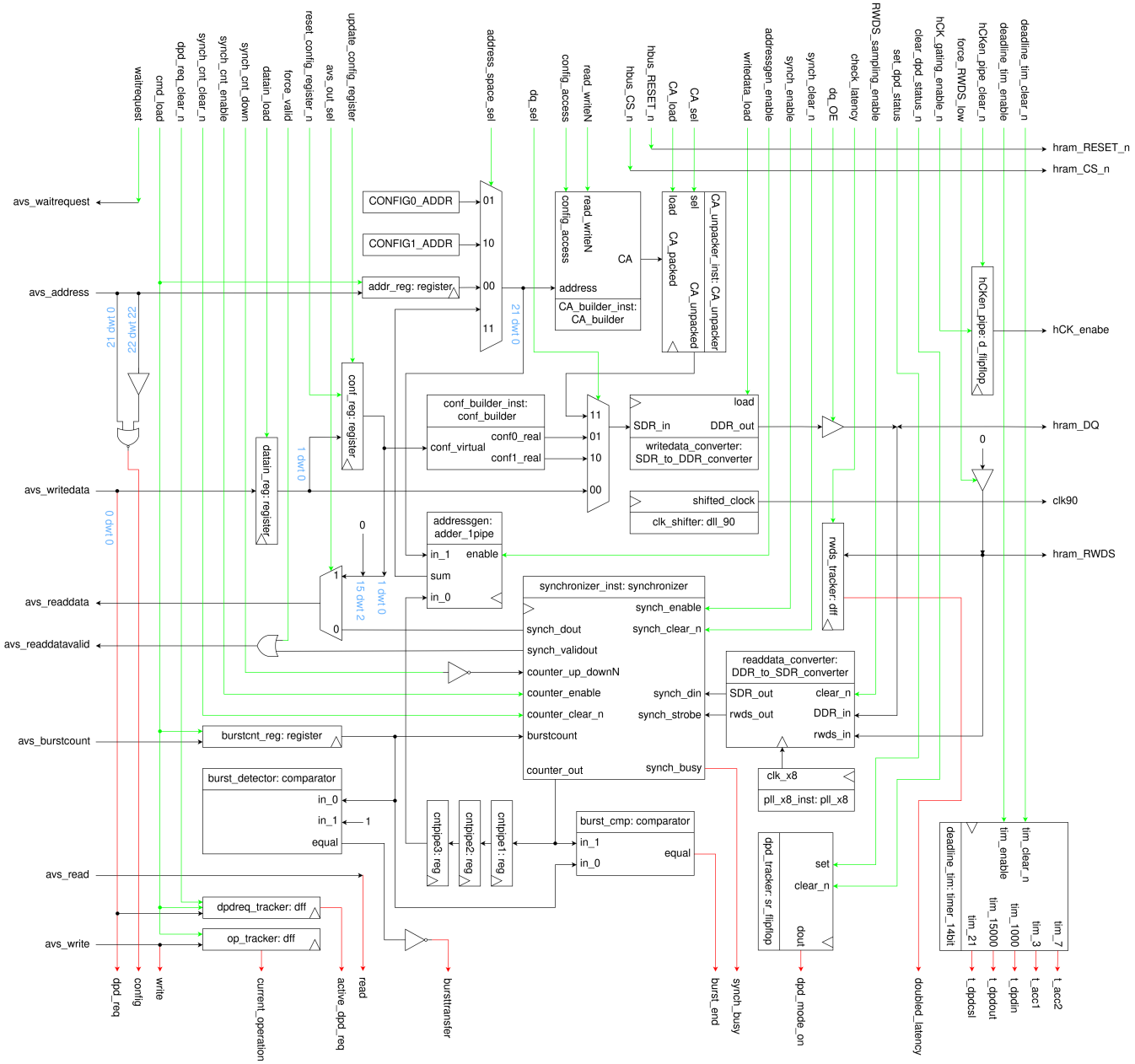

Figure 4.3: Architecture of the *avs_hram_mainconv* custom IP

TEST RESULTS

FUTURE EXTENSIONS

BIBLIOGRAPHY

[1] Massimo Ruo Roch, Maurizio Martina, *VirtLAB: a Low-Cost Platform for Electronic Lab experiments*, Sensors