June 2007

Politecnico of Torino

BRIDGE PIF / WISHBONE

Specification

Authors:
Edoardo Paone
Paolo Motto
Sergio Tota
Mario Casu

Table Of Contents

Table Of Figures

Introduction

The BRIDGE PIF / WISHBONE provides translation capabilities between two different System-on-Chip (SoC) interconnect protocols, the WISHBONE standard and the PIF protocol, so it is very useful when you have to interconnect different components in an integrated circuit.
The main difference between the two protocols is that, while the WISHBONE is in the public domain, the PIF is distributed by Tensilica with Xtensa processors and is copyrighted: so there is a large number of IP cores realized according to WISHBONE specifications, freely copied and distributed, but they can't directly communicate with PIF cores. The bridge allows to get through these limitations and connect on the same architecture components with different interfaces.
We decided to realize this device during the development of a multiprocessor architecture in Politecnico of Torino at the VLSI Lab (Electronic Departement); the processors used in the circuit design were Tensilica Xtensa microprocessors and they used the PIF protocol, so it was impossible to connect them to devices with WISHBONE interfaces, for example a WISHBONE Ethernet controller: from here the idea of a bridge in order to allow data transfers between PIF and WISHBONE devices, in a transparent way, fully compliant with the two protocols in both directions.
In the next chapter PIF and WISHBONE standards will be explained more exhaustively.

I. PROTOCOLS DESCRIPTION

I.1 WISHBONE

I.1.1 Overview

The WISHBONE System-on-Chip (SoC) Interconnect Architecture for Portable IP Cores is a portable interface for use with semiconductor IP cores. Its purpose is to foster design reuse by alleviating system-on-a-chip integration problems. This is accomplished by creating a common, logical interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user. WISHBONE itself is not an IP core, it is a specification for creating IP cores.

The WISHBONE standard is not copyrighted, and is in the public domain. It may be freely copied and distributed by any means. Furthermore, it may be used for the design and production of integrated circuit components without royalties or other financial obligations.

I.1.2 Signals Description

The WISHBONE interface of the bridge uses the following signals.

I.1.2.1 Signals Common to MASTER and SLAVE Interfaces

Name (configured width)	Direction	Description
CLK_I	Input	The clock input [CLK_I] coordinates all activities for the internal logic within the WISHBONE interconnect. All WISHBONE output signals are registered at the rising edge of [CLK_I]. All WISHBONE input signals are stable before the rising edge of [CLK_I].
DAT_I	Input	The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]).
DAT_O	Output	The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]).
RST_I	Input	The reset input [RST_I] forces the WISHBONE interface to restart. Furthermore, all internal self-starting state machines will be forced into an initial state. This signal only resets the WISHBONE interface.

I.1.2.2 MASTER signals

Name (configured width)	Direction	Description
ACK_I	Input	The acknowledge input [ACK_I], when asserted, indicates the normal termination of a bus cycle.

ADR_O()	Output	The address output array [ADR_O()] is used to pass a binary address. The higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size and granularity. For example the array size on a 32-bit data port with BYTE granularity is [ADR_O(n..2)]. In some cases (such as FIFO interfaces) the array may not be present on the interface.
CYC_O	Output	The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_O] signal is asserted during the first data transfer, and remains asserted until the last data transfer.
ERR_I	Input	The error input [ERR_I] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier
SEL_O()	Output	The select output array [SEL_O()] indicates where valid data is expected on the [DAT_I()] signal array during READ cycles, and where it is placed on the [DAT_O()] signal array during WRITE cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_O(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port.
STB_O	Output	The strobe output [STB_O] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_O()]. The SLAVE asserts either the [ACK_I], [ERR_I] or [RTY_I] signals in response to every assertion of the [STB_O] signal.
WE_O	Output	The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.
CTI_O	Output	The Cycle Type Idenfier [CTI_IO()] Address Tag provides additional information about the current cycle. The MASTER sends this information to the SLAVE.
BTE_O	Output	The Burst Type Extension [BTE_O()] Address Tag is send by the MASTER to the SLAVE to provides additional information about the current burst.

I.1.2.3 SLAVE signals

Name (configured width)	Direction	Description
ACK_O	Output	The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle.
ADR_I()	Input	The address input array [ADR_I()] is used to pass a binary address. The higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size. For example the array size on a 32-bit data port with BYTE granularity is [ADR_O(n..2)]. In some cases (such as FIFO interfaces) the array may not be present on the interface.
CYC_I	Input	The cycle input [CYC_I], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_I] signal is asserted during the first data transfer, and remains asserted until the last data transfer.
ERR_O	Output	The error output [ERR_O] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier.
SEL_I()	Input	The select input array [SEL_I()] indicates where valid data is placed on the [DAT_I()] signal array during WRITE cycles, and where it should be present on the [DAT_O()] signal array during READ cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_I(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port.
STB_I	Input	The strobe input [STB_I], when asserted, indicates that the SLAVE is selected. A SLAVE shall respond to other WISHBONE signals only when this [STB_I] is asserted (except for the [RST_I] signal which should always be responded to). The SLAVE asserts either the [ACK_O], [ERR_O] or [RTY_O] signals in response to every assertion of the [STB_I] signal.

WE_I	Input	The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.
CTI_I	Input	The Cycle Type Idenfier [CTI_IO()] Address Tag provides additional information about the current cycle. The SLAVE can use this information to prepare the response for the next cycle.
BTE_I	Input	The Burst Type Extension [BTE_O()] Address Tag is send by the MASTER to the SLAVE to provides additional information about the current burst.

I.1.3 Bus Cycles

The WISHBONE standard uses the CTI_IO(), BTE_IO() and WE_IO signals to communicate the type of transfer. Below you will find all cycle types supported by the protocol:

Read / Write Operations	
WE	**Cycle type**
0	Read cycle
1	Write cycle

Cycle Type Identifiers	
CTI_O(2:0)	**Description**
'000'	Classic cycle
'001'	Constant address burst cycle
'010'	Incrementing burst cycle
'111'	End-of-Burst

Burst Type Extension for Incrementing and Decrementing bursts	
BTE_IO(1:0)	**Description**
'00'	Linear burst
'01'	4-beat wrap burst
'10'	8-beat wrap burst
'11'	16-beat wrap burst

Now you will find a list of all cycles types provided by the WISHBONE standard:

- Classic cycle, which can support four transaction types, according to the value of WE, STB and CYC signals:

a) SINGLE READ / WRITE cycles: they perform one data transfer at a time; these are the basic cycles used to perform data transfers on the WISHBONE interconnect.
b) BLOCK READ / WRITE cycles: during BLOCK cycles, the interface basically performs SINGLE READ/WRITE cycles (called *phases*) which are combined together to form a single BLOCK cycle; the [CYC_O] signal is asserted for the duration of a BLOCK cycle: this signal can be used to request permission to access a shared resource from a local arbiter.

- Constant address burst cycle: is defined as a single cycle with multiple accesses to the same address (example: a MASTER reading a stream from a FIFO).
- Incrementing burst cycle: an incrementing burst is defined as multiple accesses to consecutive addresses; each transfer the address is incremented and the increment is dependent on the data array size: for an 8bit data array the increment is 1, for a 16bit data array the increment is 2, for a 32bit data array the increment is 4, etc. Increments can be linear or wrapped, depending on the value of BTE_IO() signal.
- End-Of-Burst: it indicates that the current cycle is the last of the current burst. The MASTER signals the slave that the burst ends after this transfer.

Some of these functions are not supported by the bridge because there is not an equivalent transfer type in the PIF protocol. These limitations will be explained in the chapter about the bridge architecture and functions.

For additional information about the WISHBONE protocol see [1].

I.2 PIF Protocol

I.2.1 Overview

The PIF protocol has evolved from the Xtensa Processor Interface description to encompass the larger scope of a system interconnect definition. Therefore, the terminology PIF (Processor Interface) keeps its literal meaning only in the historical sense. In the current context of PIF protocol definition, PIF denotes the I/O signals of any component that implements the protocol. A PIF component can be any device on the PIF interconnect network that performs in a master or slave capacity.

I.2.2 Signals Description

The PIF interface of the bridge uses the following signals (pw=configured PIF width).

I.2.2.2 MASTER signals

Name (configured width)	Direction	Description
POReqVALID	Output	Indicates that there is a valid request; all other signals prefixed with POReq are qualified by a POReqVALID assertion. It is sampled every cycle with PIReqRDY: the transfer completes when both signals are asserted.
POReqCNTL[7:0]	Output	Encodes the type, size and last transfer information for requests.
POReqADRS[31:0]	Output	Request address. During block transactions, address points to the first requested word within the block and is held stable through the transaction.
POReqDATA[pw-1:0]	Output	Data used by requests that require data during the request phase (single write and block write).
POReqDataBE[PW/8-	Output	Indicates valid bytes lanes of POReqData during requests that use POReqData, or byte lanes of

1:0]		PIRespData that are expected to be valid during responses that use PIRespData.
PIReqRDY	Input	Indicates that the slave is ready to accept requests. A request transfer completes if PIReqRDY and POReqVALID are both asserted in the same cycle.
PIRespVALID	Input	Indicates that there is a valid response. All other signals prefixed with PIResp are qualified by a PIRespVALID assertion.
PIRespCNTL[7:0]	Input	Encodes the response type and any error status.
PIRespDATA[pw-1:0]	Input	Response data; The data bus width must be equal in width to the request data bus.
PORespRDY	Output	Indicates that the master is ready to accept responses. A response transfer completes when PIRespVALID and PORespRDY are asserted in the same cycle.

I.2.2.3 SLAVE signals

Name (configured width)	Direction	Description
PIReqVALID	Input	Indicates that there is a valid request; all other signals prefixed with PIReq are qualified by a PIReqVALID assertion. It is sampled every cycle with POReqRDY: the transfer completes when both signals are asserted.
PIReqCNTL[7:0]	Input	Encodes the type, size and last transfer information for requests.
PIReqADRS[31:0]	Input	Request address. During block transactions, address points to the first requested word within the block and is held stable through the transaction.
PIReqDATA[pw-1:0]	Input	Data used by requests that require data during the request phase (single write and block write).
PIReqDataBE[PW/8-1:0]	Input	Indicates valid bytes lanes of PIReqData during requests that use PIReqData, or byte lanes of PORespData that are expected to be valid during responses that use PORespData.
POReqRDY	Output	Indicates that the slave is ready to accept requests. A request transfer completes if POReqRDY and PIReqVALID are both asserted in the same cycle.
PORespVALID	Output	Indicates that there is a valid response. All other signals prefixed with POResp are qualified by a PORespVALID assertion.
PORespCNTL[7:0]	Output	Encodes the response type and any error status.
PORespDATA[pw-1:0]	Output	Response data; The data bus width must be equal in width to the request data bus.
PIRespRDY	Input	Indicates that the master is ready to accept responses. A response transfer completes when PORespVALID and PIRespRDY are asserted in the same cycle.

I.2.3 Bus Cycles

A PIF transaction is composed of one or more distinct transfers, each of which has a hand shake and flow control mechanism, through a ReqVALID and ReqRDY or RespVALID and RespRDY signal. A PIF request transfer is considered complete when ReqVALID and ReqRDY are asserted in the same cycle; a PIF response transfer is considered complete when RespVALID and RespRDY are asserted in the same cycle.

The PIF protocol defines five request types:

- Single-data read request
- Single-data write request
- Block-read request (equivalent to the incremental-burst read transfer in WISHBONE protocol)
- Block-write request (equivalent to the incremental-burst write transfer in WISHBONE protocol)
- Read-conditional-write request

The bridge, as it will be explained in the following chapters, does not support the read-conditional-write transfer because this type is not present in the WISHBONE standard.
To perform the request and specify the type of transfer, the PIF protocol uses the ReqCNTL signal; below it's explained the meaning of the different bits:

CNTL[7:0] Signal									
BIT	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
ReqCNTL	0	0	0	0	X	X	X	1	SINGLE-READ TRANSFER
	1	0	0	0	X	X	X	1	SINGLE-WRITE TRANSFER
	0	0	0	1	X	A	B	1	BURST-READ TRANSFER
	1	0	0	1	X	A	B	0	BURST-WRITE – NOT LAST TRANSFER
	1	0	0	1	X	A	B	1	BURST-WRITE – LAST TRANSFER
RespCNTLI	0	0	X	X	X	0	0	0	RESPONSE – NOT LAST TRANSFER
	0	0	X	X	X	0	0	1	RESPONSE – LAST TRANSFER
	0	0	X	X	X	0	1	1	ADDRESS ERROR
	0	0	X	X	X	1	1	1	DATA AND BUS ERROR
	0	0	X	X	X	1	0	0	DATA ERROR – NOT LAST TRANSFER
	0	0	X	X	X	1	0	1	DATA ERROR – LAST TRANSFER

Table 1 – Information encoded in CNTL signal

The bits ReqCNTL[2:1] during requests are used to specify the block length, that is the number of bursts:

ReqCNTL[2:1]		Number of burst
0	0	2
0	1	4
1	0	8
1	1	16

Table 2 – Codification of the number of bursts

In this way when a PIF master sends a request to a slave, it uses the POReqCNTL signal to indicate the cycle type and, in case of block transfer, the number of bursts. During responses, the PIF slave can use the PORespCNTL signal to encode the error status: the bridge supports the error communication between the two interfaces but can't make difference between data and address bus errors, so it uses only the values RespCNTL[2:1]="11". For more details about PIF protocol see [2].

II. ARCHITECTURE

II.1 OVERVIEW

Figure 1 below shows general architecture of PIF / WISHBONE BRIDGE core. It consists of two main building blocks:

- PIF to WISHBONE Bridge: it has a SLAVE PIF interface on one side and MASTER WISHBONE interface on the other one, so it is used to translate the PIF MASTER requests and send them to the WISHBONE SLAVE, then to communicate the response in the other direction.

- WISHBONE to PIF Bridge: it has a SLAVE WISHBONE interface on one side and MASTER PIF interface on the other one, so it can realize

transfers between a WISHBONE MASTER device and a PIF slave, translating requests and responses from one protocol to the other one.

There is an other little block, the counter, used by the PIF to WISHBONE Bridge to increment the address in burst-transfers.

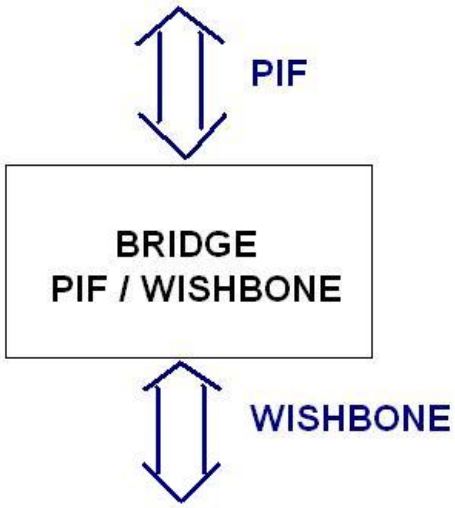


Figure 1 – Bridge PIF / WISHBONE core

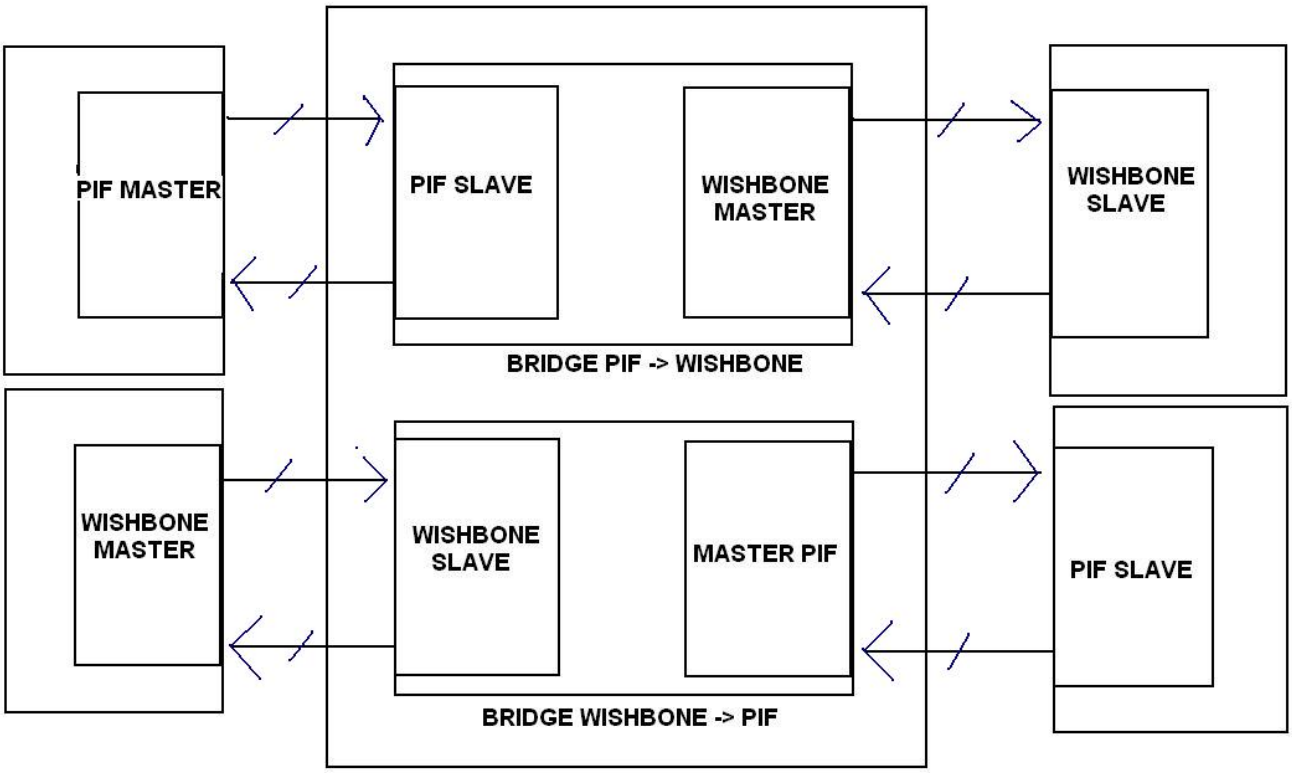


Figure 2 – Bridge PIF / WISHBONE Architecture

II.2 THE I / O PORTS

The signals used by the two protocols and described in chapter I represent the input and output ports of the bridge, because it must be able to connect master and slave devices, with both PIF or WISHBONE interfaces. The figures 3 and 4 show the different signals which represent inputs and outputs of each block. There are three signals which are common to both blocks: the clock signal (CLK), the synchronous reset signal (RST) and the asynchronous reset signal (RST_PWR), used for example on start-up of the circuit.

Note that the first letter of signals name can be an ‘M’ or ‘S’ to indicate that the signal belongs respectively to the MASTER or SLAVE interface of the bridge: this type of notation can help the user to simply and correctly connect the device.

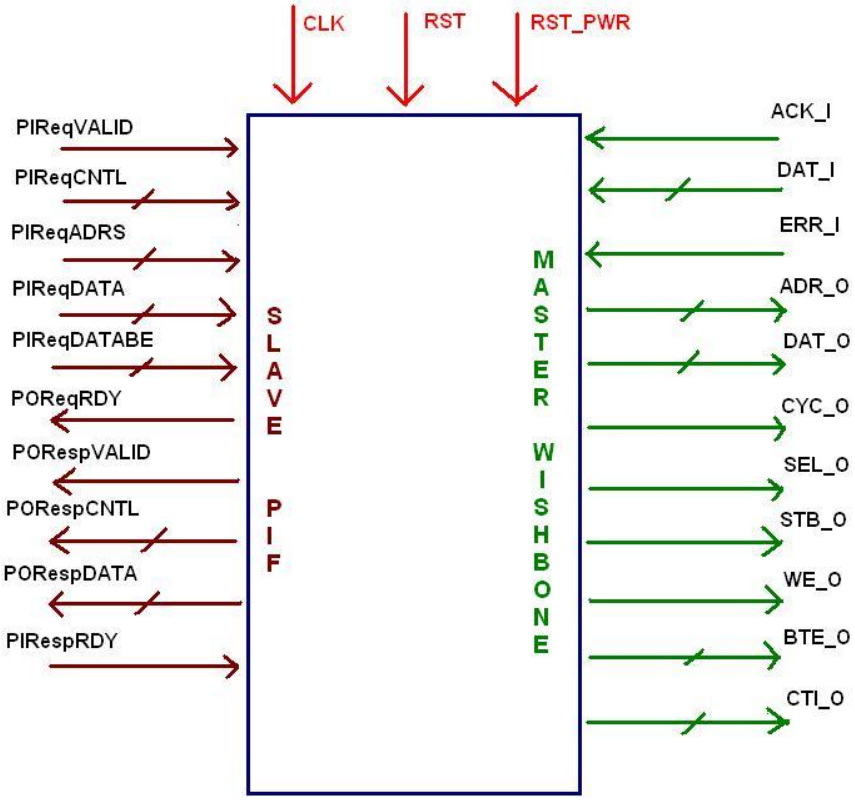


Figure 3 – PIF to WISHBONE

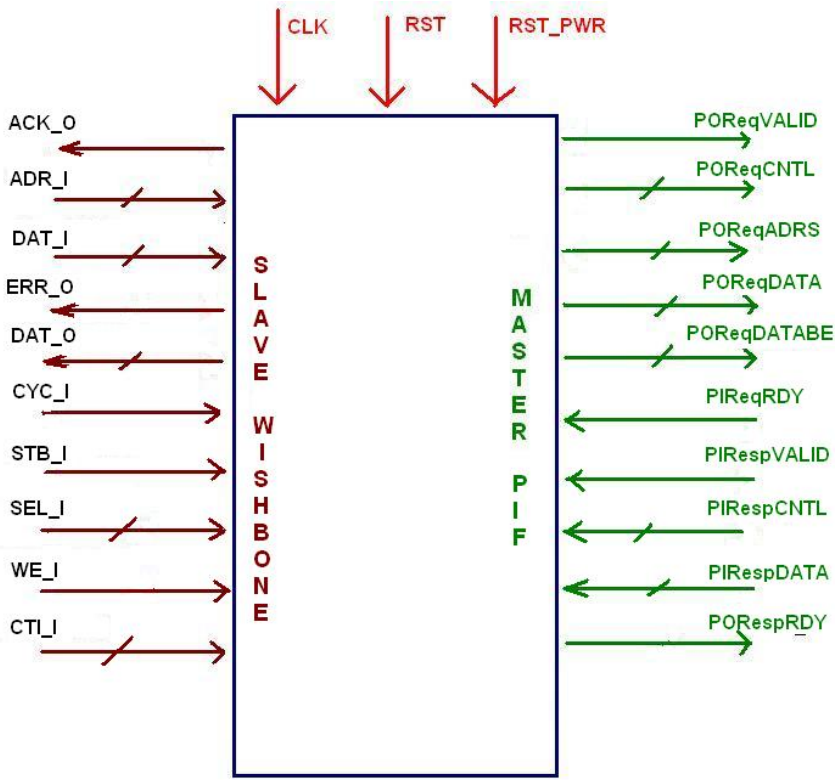


Figure 4 – WISHBONE to PIF

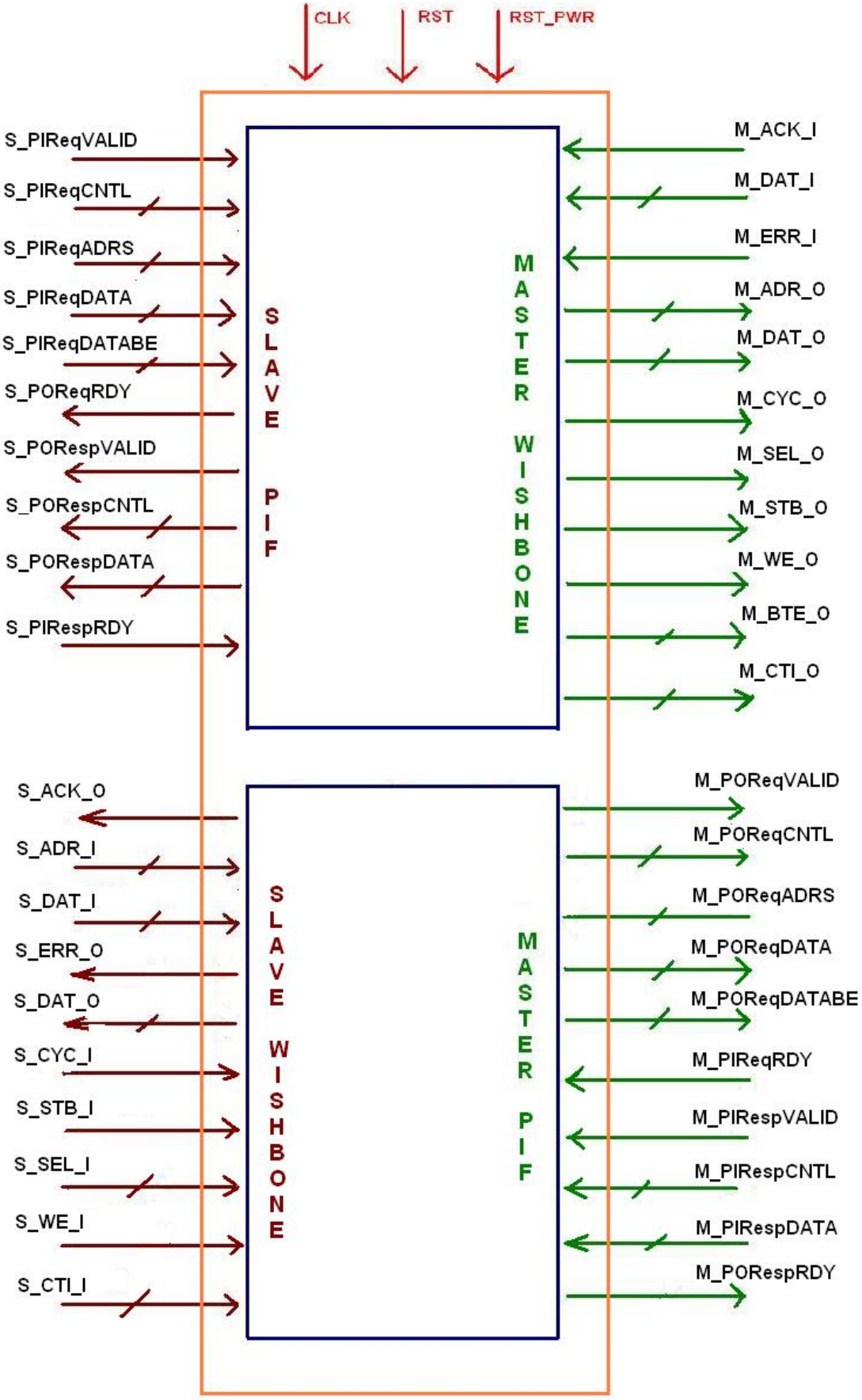


Figure 5 – Bridge I/O Signals

About the I/O signals, the two protocols allow to have signals with different length: for example the data size can have some fixed values and the same is for the address. So, to make the bridge more compliant with different IP cores, at the beginning of each VHDL source file are defined some constants which represent the bus length, the most and least significant bit and so on. Below you will find a list of these constants:

- DATA_SIZE: represents the data bus width and can assume the following values: 8, 16, 32 and 64 bits (not 128 because at the moment is not supported by WISHBONE protocol);
- ADRS_SIZE: it is the address bus width and there are not particular restrictions, depending on the memory size;
- MSB: it represents the position of the most significant bit, useful to encode information between big-endian and little-endian data representations;
- LSB: like the previous constant, it indicates the least significant bit;
- PIF_CNTL: it is the length of ReqCNTL and RespCNTL signals, fixed by the PIF protocol.

II.3 FEATURES

As explained previously, the function of this core is to allow to interconnect two devices with different interfaces (signals and protocols used): to do this the bridge must translate the signals which it receives on one side to the equivalent ones in the other protocol. In fact the PIF and WISHBONE have some characteristics in common:

1. first of all, they are both based on a full handshake protocol:
    - a PIF master begins a transaction by issuing a valid request assertion and a PIF slave indicates that is ready to accept requests by asserting a request ready signal. A request transfer completes when valid request and request-ready signals are both asserted during the same cycle. The flow of each transfer within a block transaction can be controlled with this handshake mechanism.
    - In WISHBONE protocol, the MASTER asserts [STB_O] when it is ready to transfer data. [STB_O] remains asserted until the SLAVE asserts one of the cycle terminating signals [ACK_I], [ERR_I] or [RTY_I]. At every rising edge of [CLK_I] the terminating signal is sampled. If it is asserted, then [STB_O] is negated. This gives both MASTER and SLAVE interfaces the possibility to control the rate at which data is transferred.

2. the most cycle types are provided by both the protocols; these transfer cycles in common are supported by the bridge and they are:
    - Single transfers, which can be read or write operations: read or write transactions smaller than or equal to the data-bus width that require only one data-transfer cycle are called single data transactions. Byte enables (PIReqDATABE and SEL_I) indicate which bytes should be read or written.
    - Block transfers – read or write - with incremental address: they allow block data transfers that use multiple full-width transaction cycles; so block sizes are a multiple of the bus width.

Below are reported the main differences:

1. in WISHBONE protocol there is a cycle type called block-transfer which is a little different from the block-transfer of the PIF. In fact the WISHBONE permits to do a sequence of requests, each after the other, and specify, in each transfer, the address where you want to do the read/write operation: the BLOCK cycles are modified somewhat so that these individual cycles (called *phases*) are combined together to form a single BLOCK cycle and the [CYC_O] signal is asserted for the duration of the BLOCK cycle. Instead during the PIF block transfer, the starting

address of the block transaction does not change throughout the transaction: the destination component is responsible for incrementing the address for all the words in the block: this seems to be nearer to incrementing burst cycle provided by WISHBONE standard.

2. An other difference is in the block length: the PIF protocol require that the length must be specified together with the first burst in the PIReqCNTL signal, as shown in table 2, and must have only some fixed number of bursts: 2, 4, 8 or 16; on the contrary, in WISHBONE there is not this constraint, that is to say that the length of transfer is not

specified by the master at beginning of the cycle and it can assume any value.

3. The error codification is very simple in PIF protocol, while the WISHBONE standard, to do that, requires some additional and optional signals that are not present in all IP cores.

4. In PIF protocol the data size can be 8, 16, 32, 64 or 128 bits, while in the WISHBONE the maximum size is 64 bits.

Due to these differences between the two protocols, the bridge presents some limitations in compatibility, which must be kept into account when you choose to use this bridge to interconnect two devices, otherwise the behaviour will not be that expected! So remember that:

I. The bridge supports block-transfers only with incrementing address, starting from an initial address specified during requests; the classic block-transfer of WISHBONE standard is not supported and must be divided in multiple single transfers.
II. The block length in WISHBONE to PIF direction must be fixed (8, 16, 32, 64 or 128), to do that there is a constant signal which can be modified before compiling the VHDL source code: in fact the length of the block must be communicate at the begging of the cycle in PIF standard, while this information is not required in WISHBONE; so the solution adopted is to use a constant length, equal to the most used value (statistically it is 4 or 8 bursts par block).
III. Only the presence of errors is signalized, with no error codification because this feature is not directly supported by the WISHBONE standard.

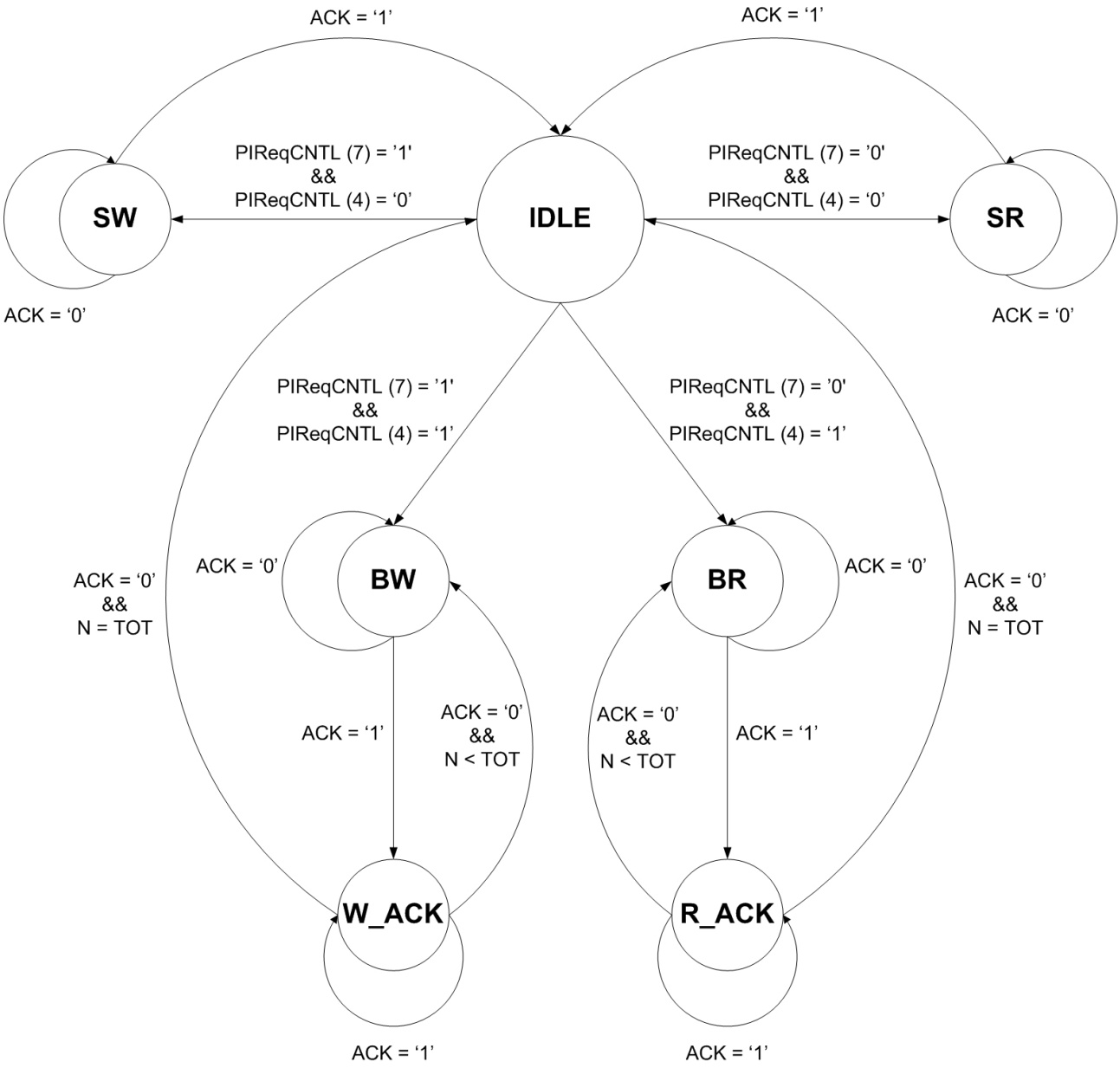
In despite of these limitations, the bridge has good performance and very simple utilisation. There would have been a solution to the first problem, with a queue buffer: this solution was inefficient because of retard times and could have presented problems because of memory size and loss of data bursts.

II.4 OPERATION

The problem about translating one protocol into the other was essentially in synchronizing the handshake signals, in particular during block transfers. I decided to implement these functions with a Finite State Machine (FSM):

a) I found the main states, corresponding to different phases of the transfer cycles;
b) for each state I selected the activation signals, used to decide how to go from one state to another;
c) each state corresponds to a certain phase, so it is characterized by certain values of output signals.

As shown in Figure 5, the architecture of the bridge has two main blocks, each one corresponding to a FSM. In the following pages the operation will be explained with a state diagram.



State :
IDLE
SR : Single Read
SW : Single Write
BR : Block Read
BW : Block Write
R_ACK : Response to ACK in Block Read
W_ACK: Response to ACK in Block Write

Signals :
N : Current Number of Trasfers
TOT : Total Number of Transfers
ACK : Wishbone Acknowledge
PIReqCNTL : PIF Request Control

Figure 6 – PIF to WISHBONE FSM

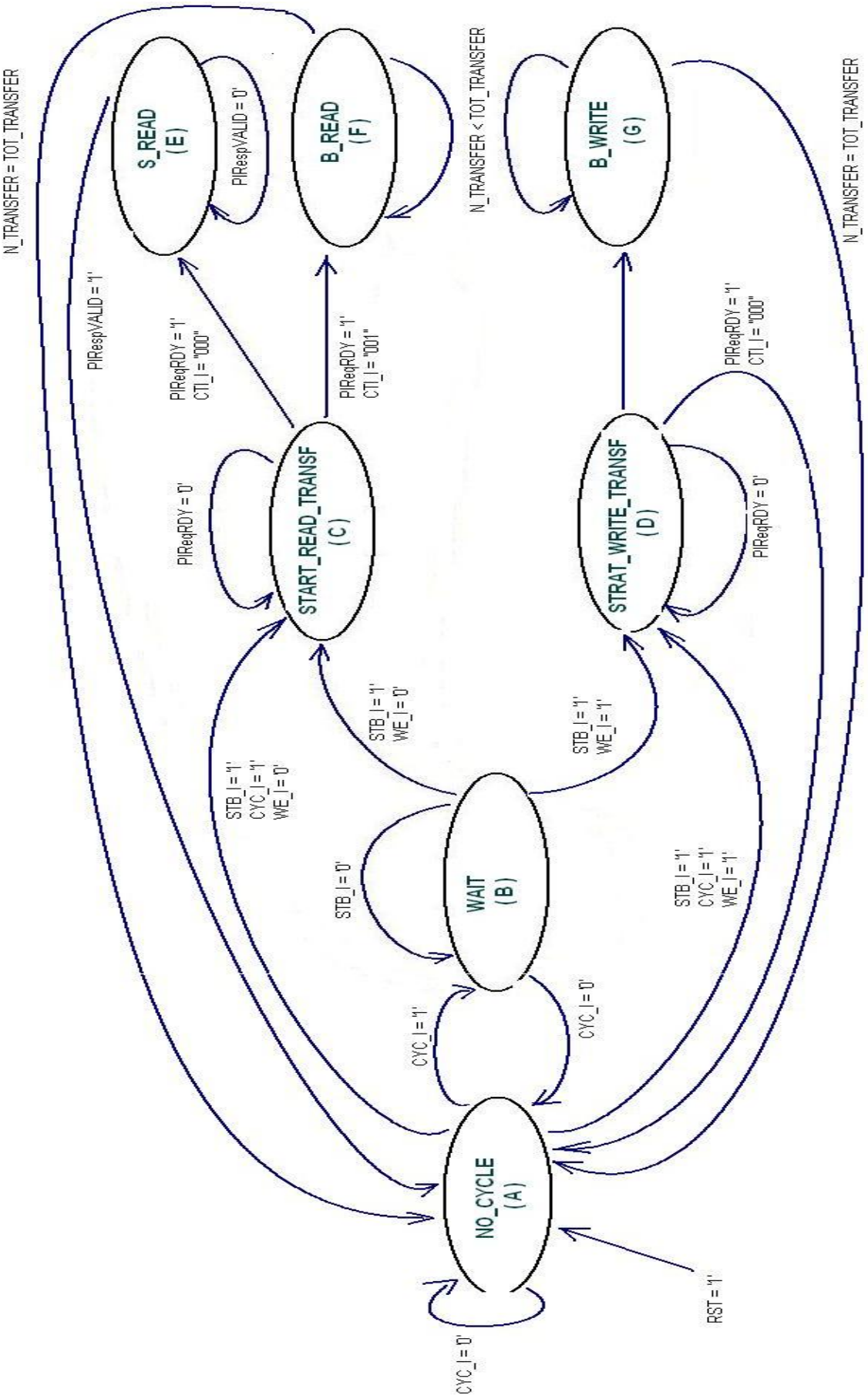


Figure 7 – WISHBONE to PIF FSM

To realize the design of the two FSM, I studied the two protocols to understand the operation of each one and to find out a way to translate the different cycle types from one side to the other. In each state the bridge controls the output signals, the main difficult was in synchronising the transitions from one state to the following – note that in general there is more than one possible choice, depending on activation input signals. The VHDL description uses three main processes:

1. **State_Register:** this is a sequential process which, on the clock rise edge, assigns the new value to signal **state**, essentially **next_state** if **RST='0'** otherwise it puts the bridge into the initial state, that is state A.
2. **Next_State_Function:** it's a combinational process; depending on the current state and the activation signals, it assigns the correct value to **next_state**, so it decides what the bridge would do the next clock period.
3. **Output_Function:** it's a combinational process which controls almost all output signals depending on inputs and current state.

After that I described in VHDL the FSM corresponding to the bridge, I used Xilinx ModelSIM to compile the source code and test it. In particular, it was very useful the waveform simulation tool, because I could compare the behaviour of signals of the bridge with that expected, which is described in the diagrams in the next chapter. To do this simulation, I wrote a testbench in VHDL which is reported in chapter V.

III. DIAGRAMS

III.1 WISHBONE to PIF – Single Write Cycle

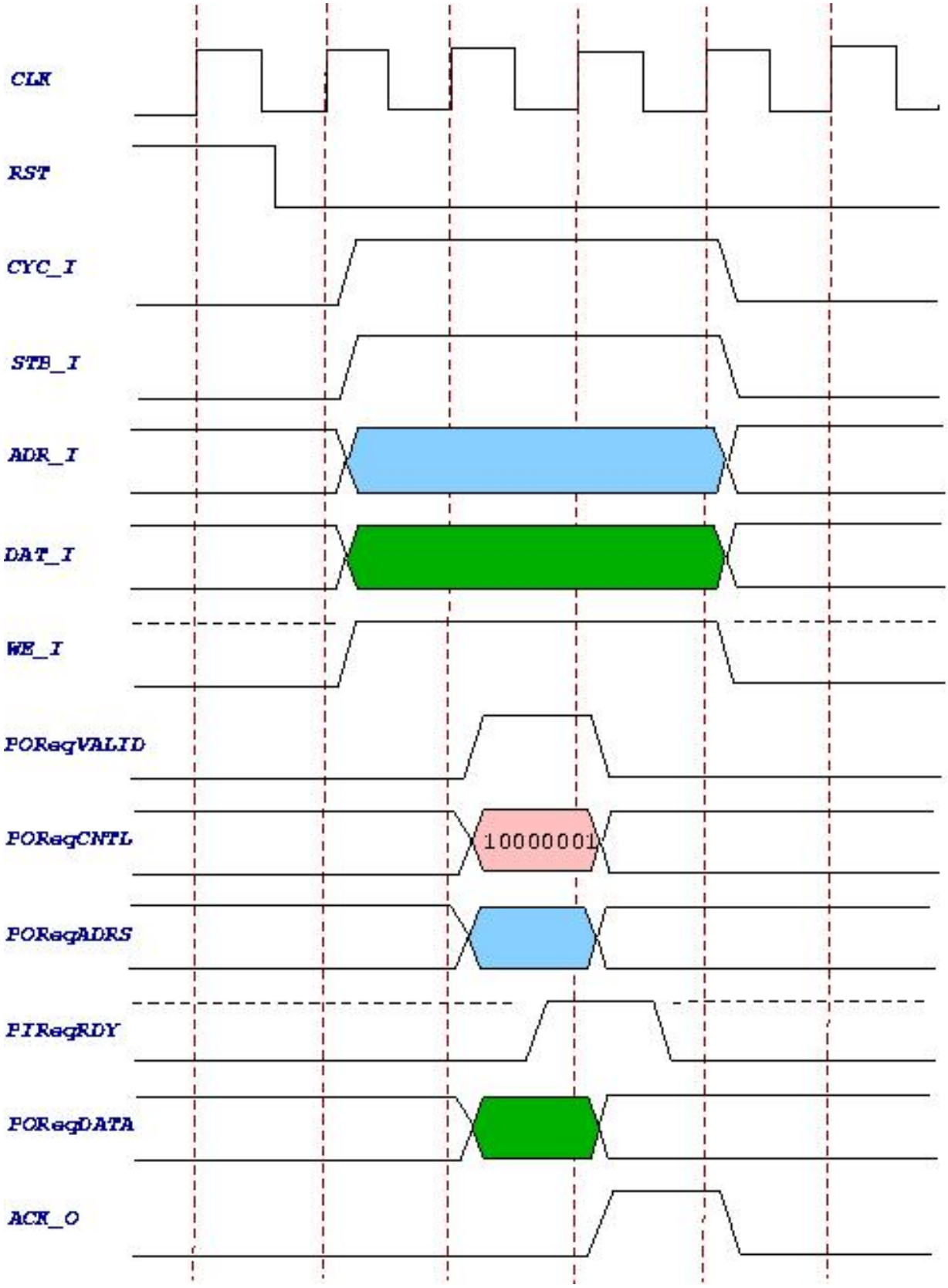


Figure 8 – WISHBONE to PIF single write cycle

III.2 WISHBONE to PIF – Block Write Cycle

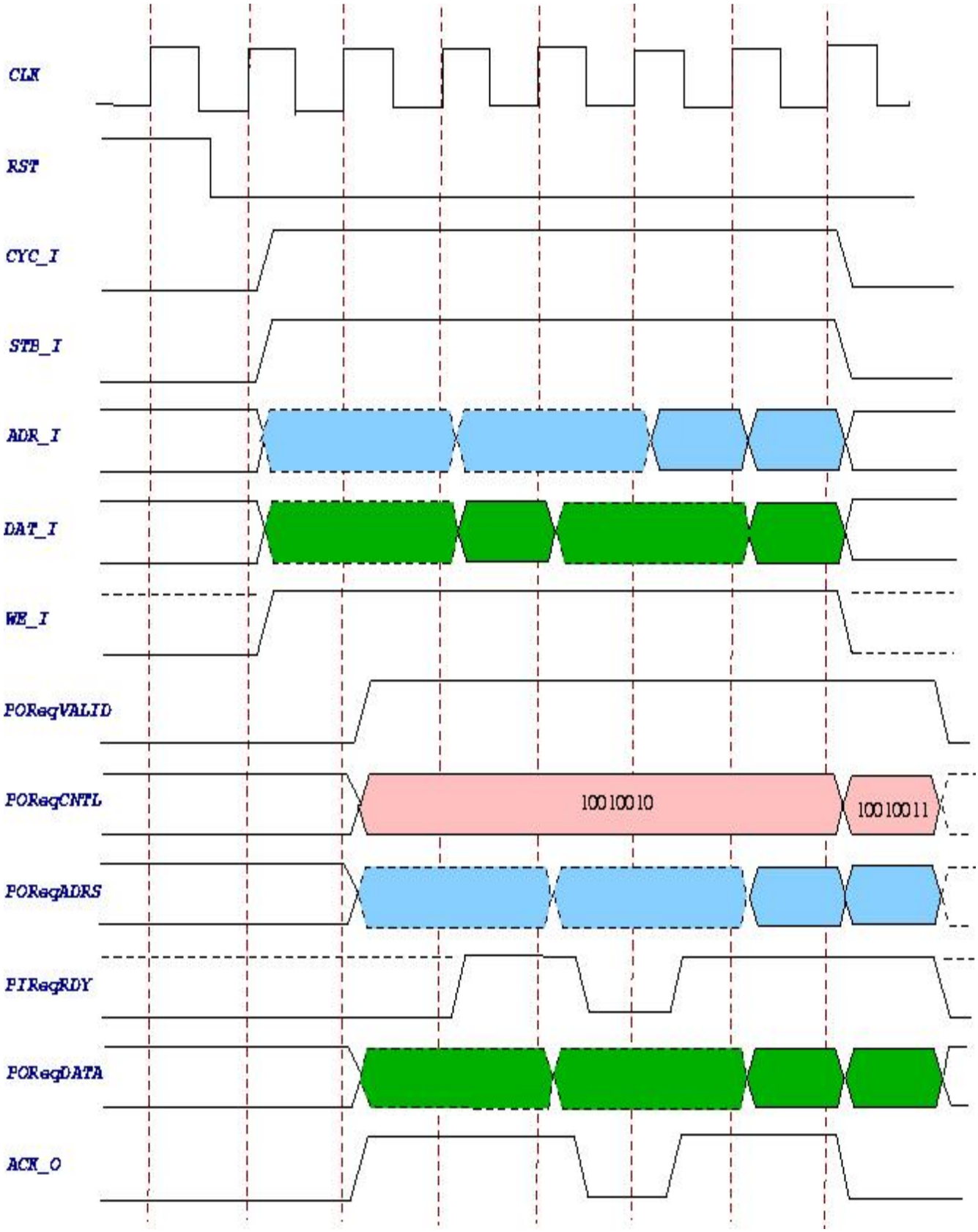


Figure 9 – WISHBONE to PIF block write cycle

III.3 PIF to WISHBONE – Single Read Cycle

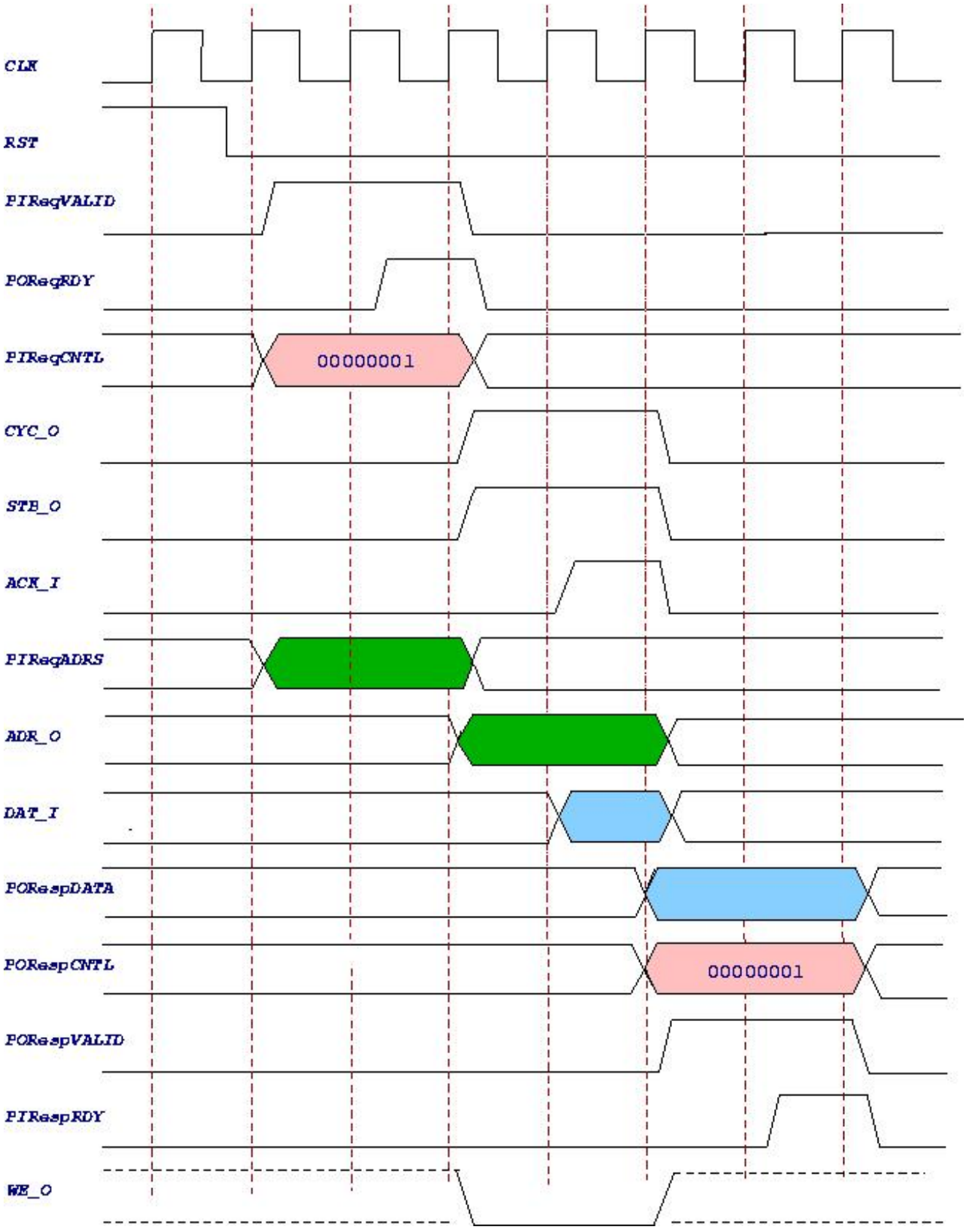


Figure 10 – PIF to WISHBONE – single read cycle

III.4 PIF to WISHBONE – Block Read Cycle

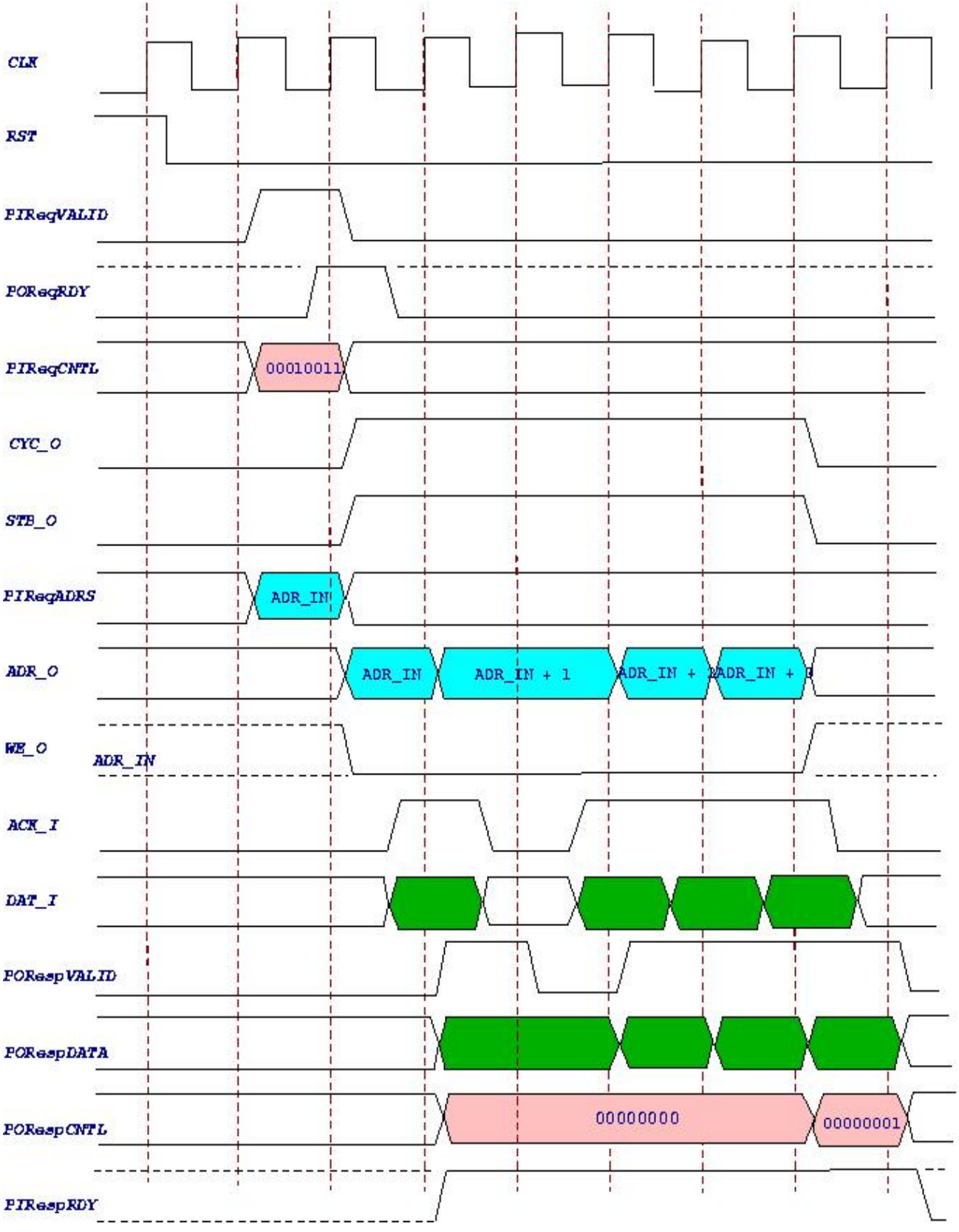


Figure 11 – PIF to WISHBONE – block read cycle

IV. FPGA IMPLEMENTATION

IV.1 SYNTHESIS

After describing the bridge in VHDL, the second step was the synthesis and implementation of the design on an FPGA: to do this work I used the Xilinx Integrated Software Environment (ISE). I loaded my project into this program and I proceed with the synthesis on a Xilinx Virtex4 FPGA: the result of this operation was a RTL schematic of the bridge and other information about the utilisation of logic components, reported below.

Figure 12 – Bridge RTL schematic

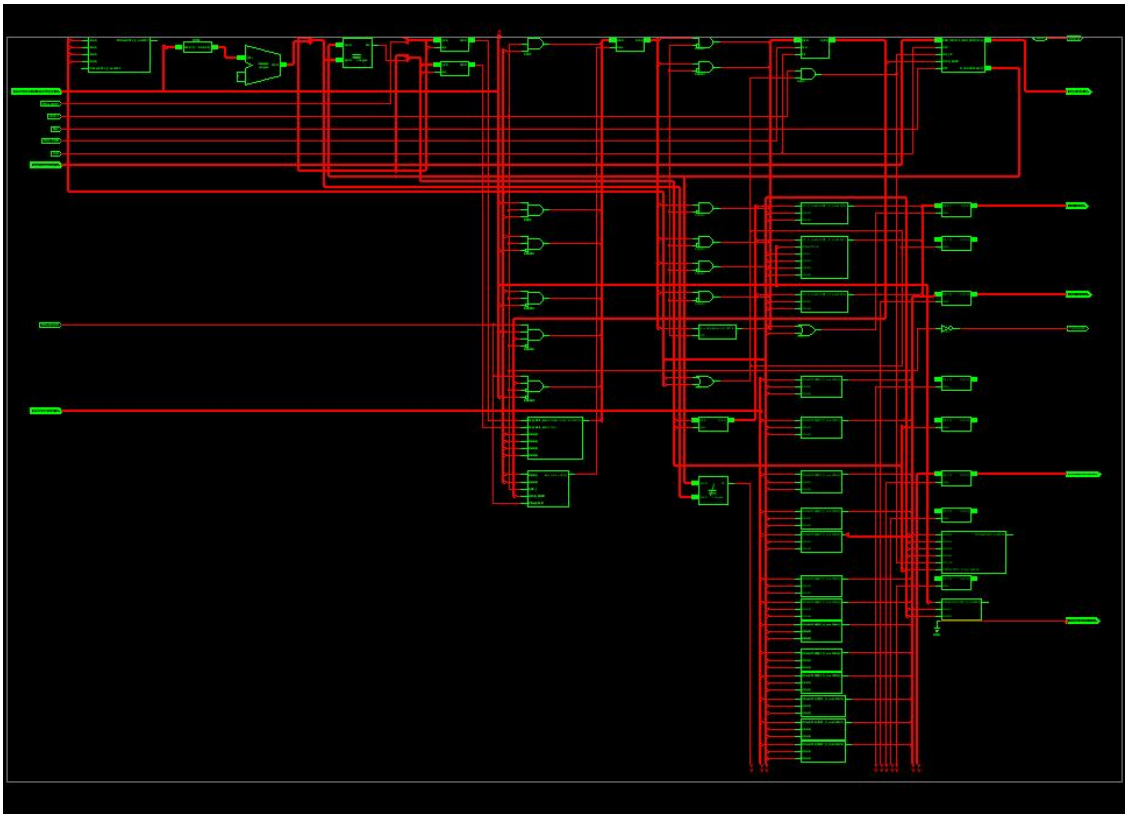


Figure 13 – PIF to WISHBONE RTL schematic

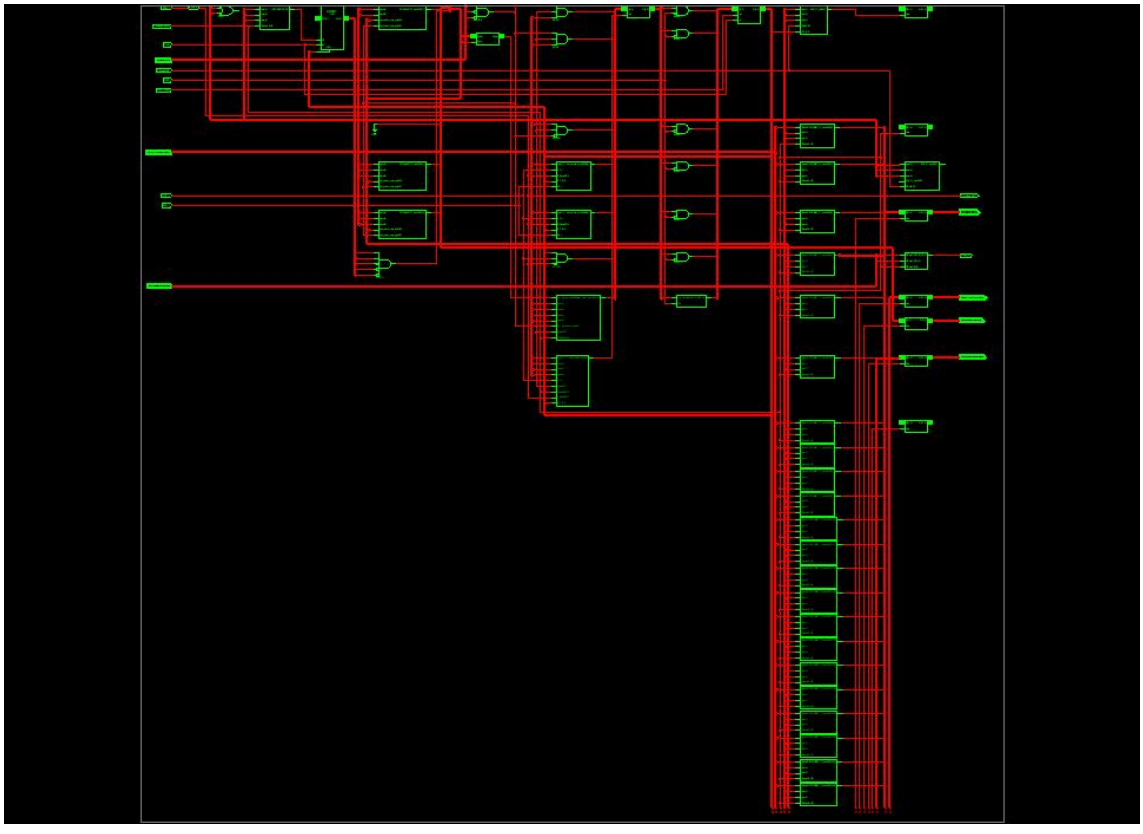


Figure 14 – WISHBONE to PIF RTL schematic

Below you find the design summary reported by Xilinx ISE after synthesis on Virtex4 FPGA:

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	72	178,176	1%
Number used as Flip Flops	56		
Number used as Latches	16		
Number of 4 input LUTs	274	178,176	1%
Logic Distribution			
Number of occupied Slices	156	89,088	1%
Number of Slices containing only related logic	156	156	100%
Number of Slices containing unrelated logic	0	156	0%
Total Number of 4 input LUTs	303	178,176	1%
Number used as logic	274		
Number used as a route-thru	29		
Number of bonded IOBs	451	960	46%
Number of BUFG/BUFGCTRLs	8	32	25%
Number used as BUFGs	8		
Number used as BUFGCTRLs	0		
Total equivalent gate count for design	3,250		
Additional JTAG gate count for IOBs	21,648		

IV.2 IMPLEMENTATION

After implementation, I used both Xilin ModelSIM and Xilinx ISE to do post-map and post-route simulation. I found some timing errors and I had to modify the VHDL description in order to take into consideration delays introduced by

the physical layer. After that, I did the simulation using the same testbench and the result was satisfying.

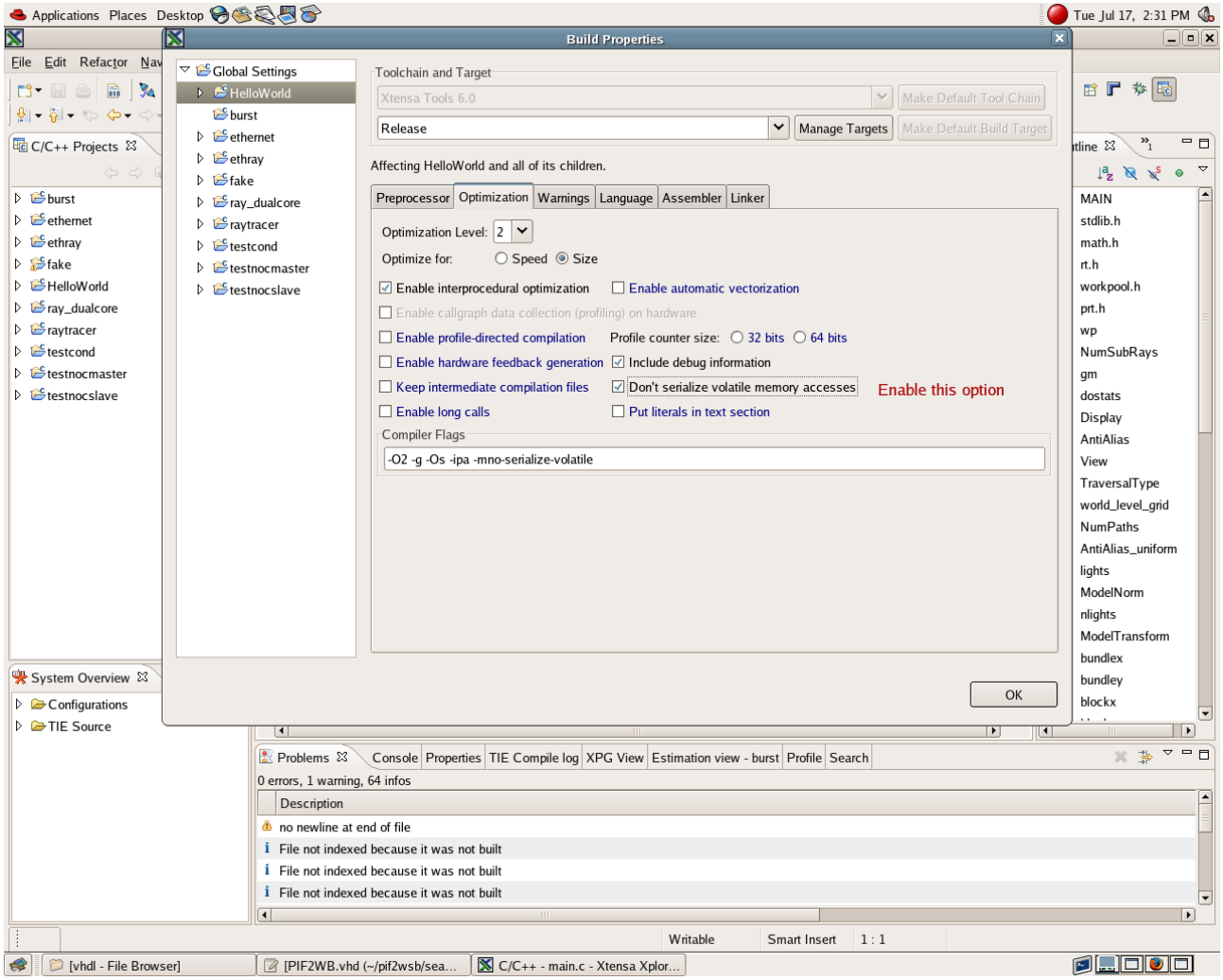
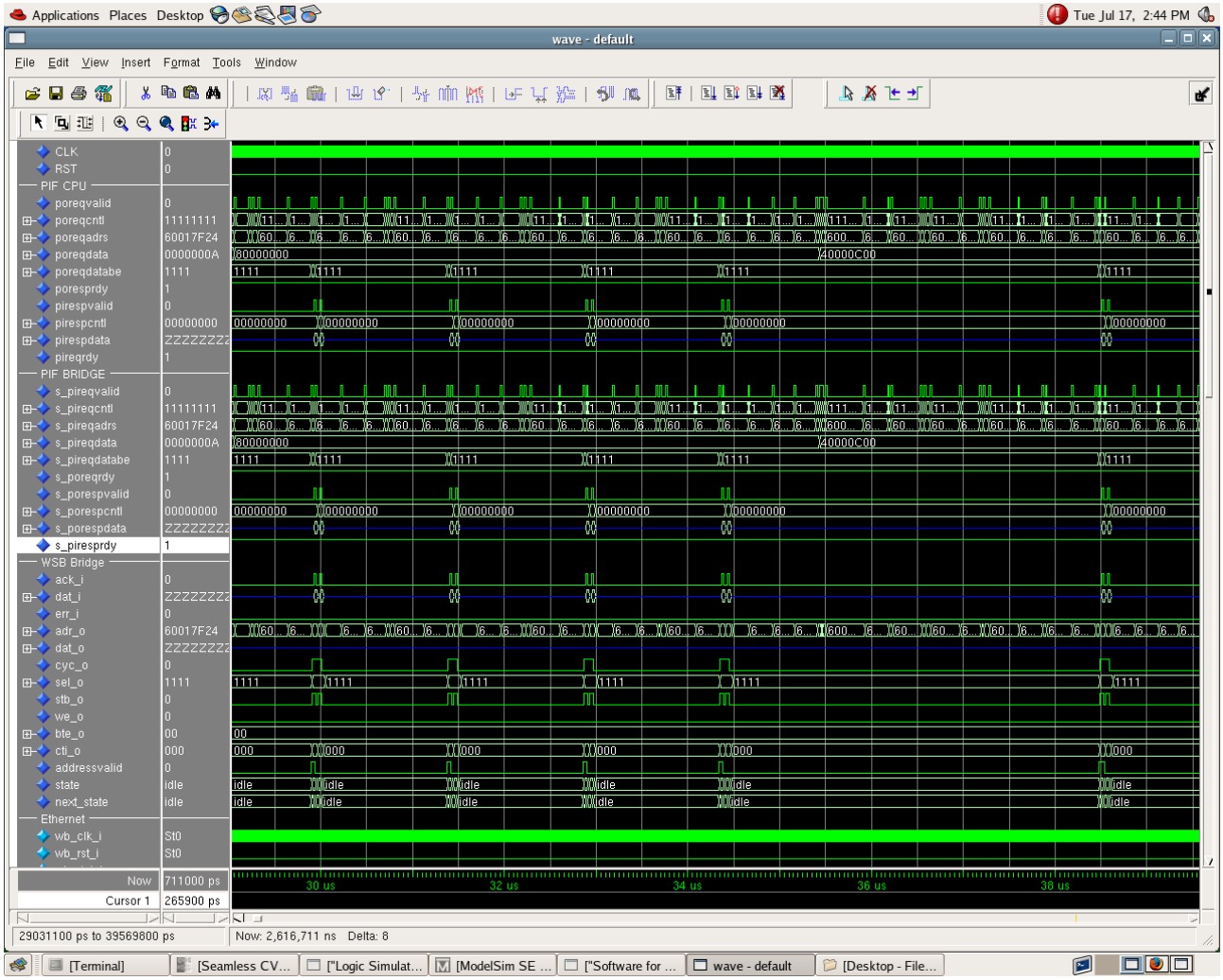


Figure 15 – Compile Flags for Block read / write


Applications Places Desktop
Tue Jul 17, 2:44 PM
wave - default
File Edit View Insert Format Tools Window
CLK
RST
PIF CPU
poreqvalid
poreqcntl
poreqadrs
poreqdata
poreqdatabe
poresprdy
pirespvalid
pirespcntl
pirespdata
pireqrdy
PIF BRIDGE
s_pireqvalid
s_pireqcntl
s_pireqadrs
s_pireqdata
s_pireqdatabe
s_poreqrdy
s_porespvalid
s_porespcntl
s_porespdata
s_piresprdy
WSB Bridge
ack_i
dat_i
err_i
adr_o
dat_o
cyc_o
sel_o
stb_o
we_o
bte_o
cti_o
addressvalid
state
next_state
Ethernet
wb_clk_i
wb_rst_i
Now
Cursor 1
30 us
32 us
34 us
36 us
38 us
29031100 ps to 39569800 ps
Now: 2,616,711 ns Delta: 8
[Terminal]
[Seamless CV...
["Logic Simulat...
[ModelSim SE ...
["Software for ...
wave - default
[Desktop - File...

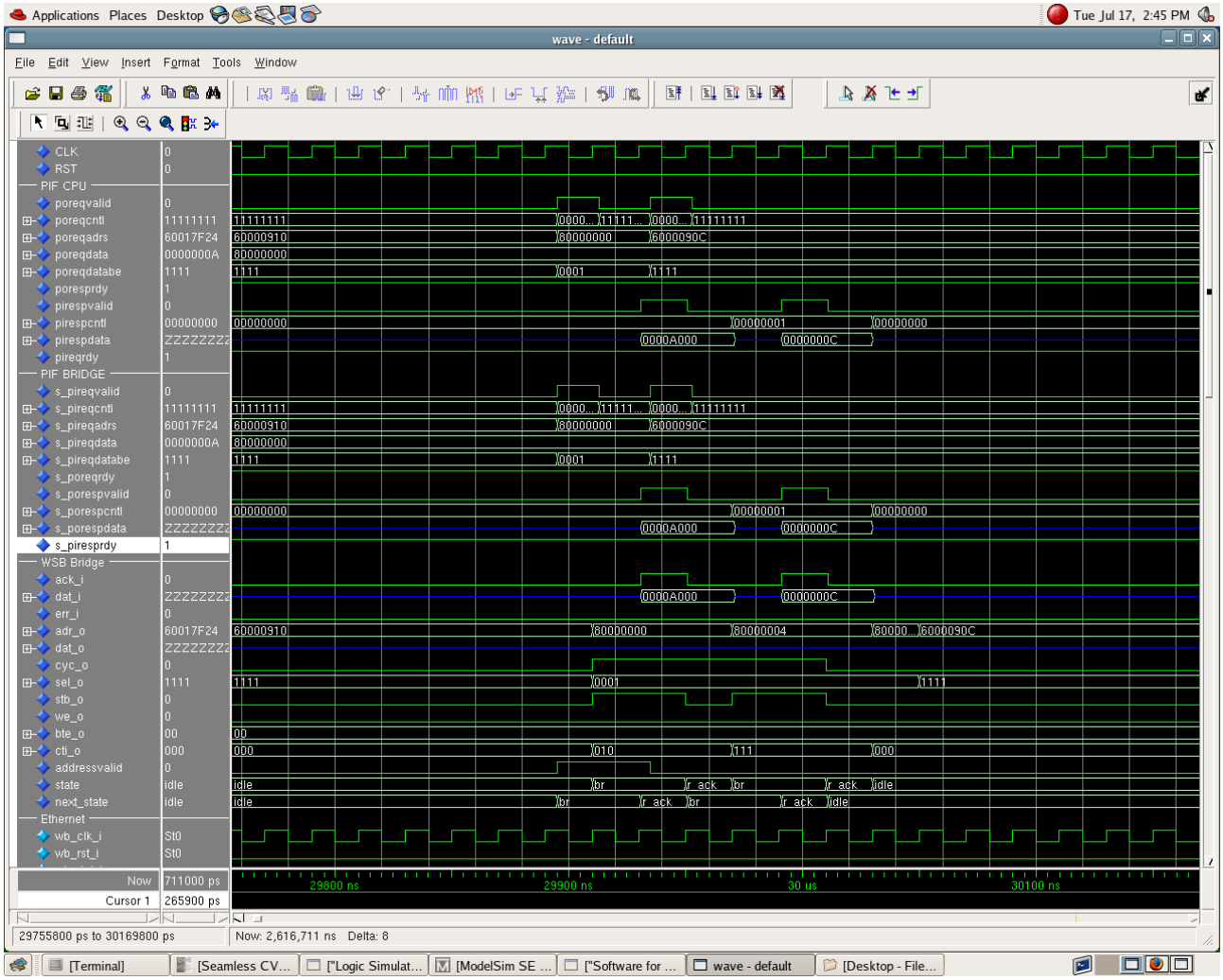


Figure 16 and 17 – Seamless Hardware & Software Co – simulation

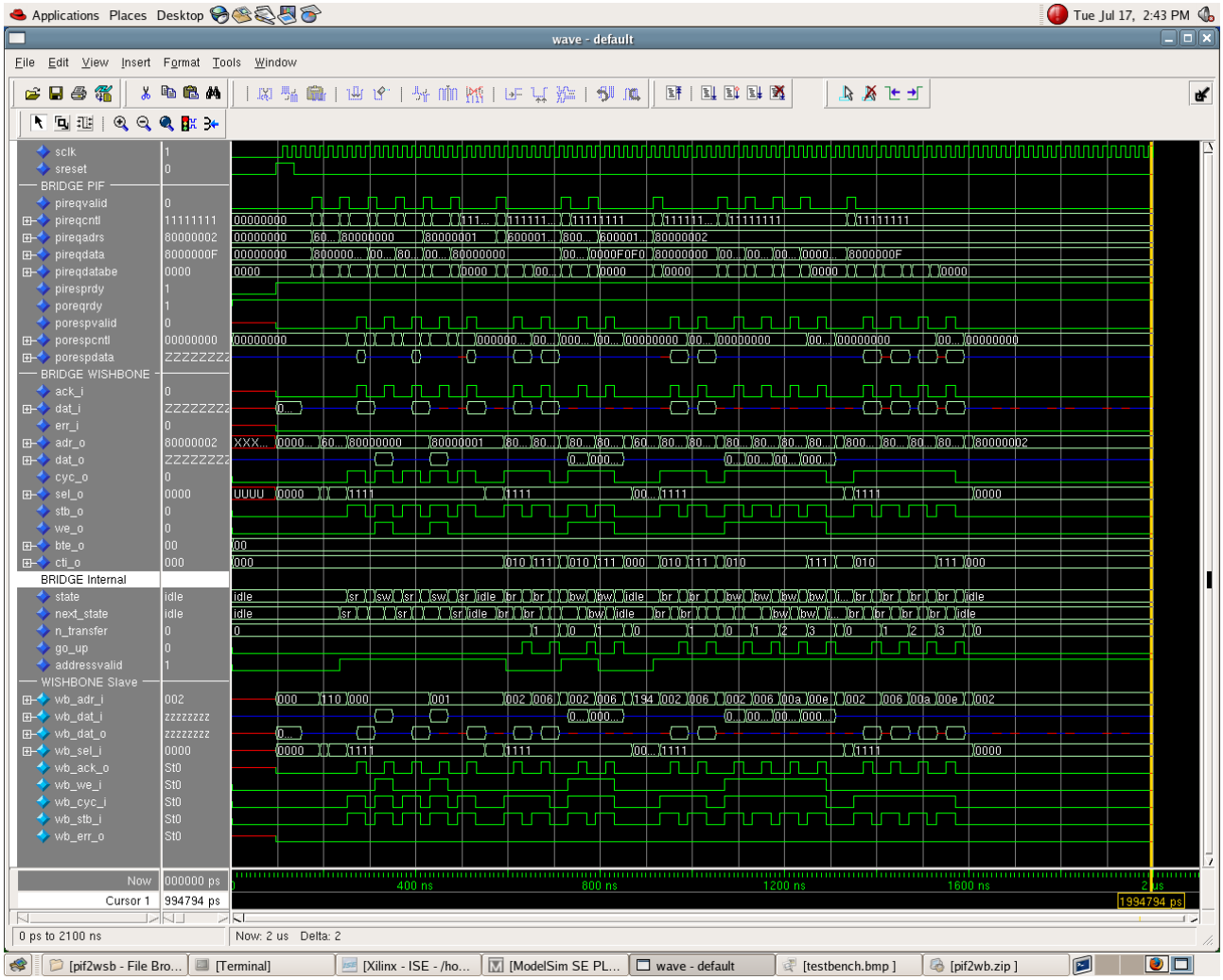


Figure 18 – Testbench output waveform

Bibliography

[1] WISHBONE Specification Revision B.3, see the OpenCores website
http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf

[2] Xtensa Processor Interface Protocol (PIF), see the Tensilica website
http://www.tensilica.com