FINAL PROJECT

On-chip bus standards in a broadcast architecture

Field: Electronic Engineering
Author: Josep-Oriol Romaguera i Restudis
Supervisor: Lauri Koskinen
May 2012
Aalto University - School of Electrical Engineering
Department of Micro- and Nanosciences (Electronic Circuit Design unit)

Bachelor’s Thesis

Author: Josep-Oriol Romaguera i Restudis

Title: On-chip bus standards in a broadcast architecture

Date: 21.05.2012 Language: English Number of Pages: 48

Supervisor: Lauri Koskinen

**Keywords:** Broadcast communication, AMBA, wire model, wire delay, VHDL.
Contents

Acknowledgment........................................................................................................................................................................4
List of abbreviations and symbols..................................................................................................................................................5
Abstract.............................................................................................................................................................................................................6
1. Introduction................................................................................................................................................................................................7
   1.1 Objective and Scope.................................................................................................................................................................7
   1.2 Outline ................................................................................................................................................................................................8
2. Theory......................................................................................................................................................................................................9
   2.1 AMBA Bus ..................................................................................................................................................................................................9
   2.2 CAN Bus ................................................................................................................................................................................................15
3. System approach................................................................................................................................................................................................17
   3.1 Specifications ................................................................................................................................................................................................17
   3.2 Design alternatives ......................................................................................................................................................................18
   3.3 System block diagram ...............................................................................................................................................................21
4. Design....................................................................................................................................................................................................23
   4.1 AMBA AHB system (VHDL)........................................................................................................................................................23
      4.1.1 Master................................................................................................................................................................................................23
      4.1.2 Slave ................................................................................................................................................................................................26
      4.1.3 Arbiter ................................................................................................................................................................................................27
      4.1.4 Decoder ................................................................................................................................................................................................30
   4.2 Wire model................................................................................................................................................................................................31
      4.2.1 Calculation of wire parameters........................................................................................................................................31
      4.2.2 Wire delay and energy per bit ...........................................................................................................................................33
      4.2.3 Wire delay model in VHDL ...........................................................................................................................................38
5. Verification................................................................................................................................................................................................42
6. Conclusions..................................................................................................................................................................................................45
7. References................................................................................................................................................................................................47
Appendix A. AHB Signals.....................................................................................................................................................................I
Appendix B. VHDL Code........................................................................................................................................................................IV
Acknowledgment

I would like to thank my supervisor, Lauri Koskinen, for giving me the opportunity to work in a project and providing inspiration and help when needed.

In addition, I want to thank the student service personal, especially Nina Huovinen, for all the answered mails and make my arrival easier.

Last but not least, I would like to thank my family and, mainly, my girlfriend, for the support given during the last 8 months and “allow me” to do the final project abroad.
# List of abbreviations and symbols

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA</td>
<td>Advanced Microcontroller Bus Architecture</td>
</tr>
<tr>
<td>AHB</td>
<td>Advanced High-performance Bus</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>VHDL</td>
<td>Combination of the abbreviations “VHSIC” and “HDL”</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{\square}$</td>
<td>Sheet resistance</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Electrical resistivity</td>
</tr>
<tr>
<td>$k = \varepsilon_r$</td>
<td>Relative permittivity</td>
</tr>
<tr>
<td>$\varepsilon_{ox}$</td>
<td>Oxide permittivity</td>
</tr>
<tr>
<td>$\varepsilon_0$</td>
<td>Vacuum permittivity ($8.854\ldots \times 10^{-12}$ F/m)</td>
</tr>
<tr>
<td>$T_{RC}$</td>
<td>Delay of a wire (using RC lumped model)</td>
</tr>
<tr>
<td>$r_{wire}$</td>
<td>Wire resistance per millimetre</td>
</tr>
<tr>
<td>$c_{wire}$</td>
<td>Wire capacitance per millimetre</td>
</tr>
<tr>
<td>$D_{total}$</td>
<td>Total delay of a 1mm wire</td>
</tr>
<tr>
<td>$D_{repeater}$</td>
<td>Repeater delay</td>
</tr>
<tr>
<td>$N_{opt}$</td>
<td>Optimal sections of the wire (to minimize the delay)</td>
</tr>
<tr>
<td>$C_L$</td>
<td>Load capacitance</td>
</tr>
<tr>
<td>$V$</td>
<td>Supply voltage</td>
</tr>
<tr>
<td>$E_b$</td>
<td>Total bit energy per useful bit and per millimetre</td>
</tr>
<tr>
<td>$E_b'$</td>
<td>Bit energy per useful bit and per millimetre (without repeaters)</td>
</tr>
<tr>
<td>$E_{buff}$</td>
<td>Repeater energy per useful bit</td>
</tr>
</tbody>
</table>
Abstract

This final project has been developed in the School of Electrical Engineering (in Aalto University) and its main goal is to value whether AMBA (Advanced Microcontroller Bus Architecture) is a good option to develop an on-chip broadcast architecture.

To carry it out, various alternatives are taken in account: use full AMBA (with all its signals), use CAN (Controller Area Network) bus instead of AMBA, and use modified AMBA (using only the signals that will be used). Once one alternative is chosen, the implementation is done, using VHDL (Very high speed integrated circuit - Hardware Description Language).

Afterwards, wire delay per millimetre is calculated (with and without buffering). With the optimization, a 40.7% of improvement is obtained, with a delay of $68.465 \text{ ps/mm}$. In addition, a delay model in VHDL is designed and implemented.

Finally, energy per bit is also calculated and compared with a point-to-point architecture system. Obviously, the performed design consumes more energy per bit ($0.558 \text{ pJ/bit-mm}$) than a point-to-point system. Nevertheless, it is important to highlight that a point-to-point communication needs many wires.
1. Introduction

This chapter shows the main goals of the thesis and describes, chapter by chapter, the contents of each one.

1.1 Objective and Scope

The first goal of this project is to implement an on-chip broadcast architecture, using VHDL (Very high speed integrated circuit - Hardware Description Language). To carry it out, a kind of shared bus, such as AMBA (Advanced Microcontroller Bus Architecture) or CAN (Controller Area Network), must be used. Therefore, before doing any VHDL description, various design alternatives have to be set out and evaluated. Furthermore, it is important to point out that this architecture must be designed for a huge number of modules, which can transmit between each other. However, the design starts with a reduced number of modules (eight), allowing an easier explanation and implementation. After that, and once the eight modules system has been simulated, the design is extended to N modules, where N may take any value.

Following that, the second goal is to model the wire delay of the interconnections between these modules. Throughout most of the past history of integrated circuits, on-chip interconnect wires were only considered in special cases. With the introduction of deep-submicron semiconductor technologies, the situation changed rapidly. The parasitic effects introduced by the wires show a scaling behaviour that differs from the active devices such as transistors, and tend to gain in importance as device dimensions are reduced and circuit speed is increased. In fact, they start to dominate some of the relevant metrics of signal integrated circuits such as speed, energy-consumption and cost. Therefore, a careful and in-depth analysis of the role and the behaviour of the interconnect wire in a semiconductor technology is not only desirable, but even essential. Consequently, this part of the thesis consists of finding the wire parameters (such as resistance and capacitance per millimetre), the delay per millimetre, and energy consumed per useful bit and per millimetre. Note that in the last part (energy per bit), the placement of modules must be discussed in order to obtain minimum distances between modules.

Finally, after to model the delay using VHDL, the system is simulated using ModelSIM. At this point, the behaviour of the system might be checked. In addition, a comparison between non-delayed and delayed system should be carried out, in order to obtain conclusions about delay model, and determine over how many modules the system can transfer.
1.2 Outline

The main contents of this document is split in five chapters:

- **Chapter 2:** some lines of theory are introduced in this chapter, which deals with AMBA bus, and briefly CAN bus.

- **Chapter 3:** the system approach is shown in this chapter. Specially, the specifications of the design may be seen, as well as the design alternatives, and the block diagram of the chosen alternative.

- **Chapter 4:** all design process may be seen in this chapter, which is divided in two sections. The first one shows all the VHDL design of the chosen alternative. On the other hand, the second section deals with the wire parameters, such as resistance and capacitance, the calculation of the wire delay and the energy per bit and, finally, it shows the wire model and its VHDL implementation.

- **Chapter 5:** the main goal of this chapter is to verify that the system works satisfactory. To carry it out, some commented simulations are shown.

- **Chapter 6:** finally, last chapter presents some conclusions about the design.
2. Theory

This chapter presents theory that helps to understand this document. Basically, it shows how AMBA bus works and, more briefly, how CAN bus works.

2.1 AMBA Bus

AMBA (Advanced Microcontroller Bus Architecture) is a specification introduced by ARM Ltd in 1996 and it is used as on-chip bus in system-on-chip (SoC) designs. The first AMBA buses were “Advanced System Bus” (ASB) and “Advanced Peripheral Bus” (APB). Then, AHB (Advanced High-performance Bus) protocol was introduced, completing the AMBA2 version.

In 2003, ARM introduced the third generation, AMBA3, and later AMBA4. These generations have also its own protocols, but they do not be explained because they are not the aim of this thesis. In fact, AMBA2 is enough because it is the simplest one [1].

Regarding to AMBA 2, it is important to point out that it has three different buses. Their specifications are shown below [1] [2]:

1) Advanced High-performance Bus (AHB)
   b. High clock frequency.
   c. It is the main bus. Supports efficient connections to processors, memories, etc.

2) Advanced System Bus (ASB)
   a. It is also for high-performance.
   b. It is an alternative to the AHB bus when the high-performance features of AHB are not required.

3) Advanced Peripheral Bus (APB)
   a. It is for low-power peripherals.
   b. Interface less complex than AHB and ASB.
   c. AHB/ASB – APB bridge is required to connect an AHB/ASB bus with APB bus.

A typical AMBA 2 system is illustrated in Figure 1.
In this project, only AMBA AHB protocol is used because it is enough for our goals. APB is not needed because it is oriented to peripherals devices, and ASB is similar to AHB. However, the last one has more performances hence AHB will be used.

Focusing now to AHB bus, it is important to highlight that it is a bus designed to support the requirements of the high-performance designs, high operating frequencies, including among other features [1] [3]:

- Burst transfer.
- Split transactions.
- Single cycle bus master handover.
- Single clock edge operation.
- Non-tristate implementation.
- Wider data bus configurations (64/128 bits).

In the following lines, a brief introduction to AMBA AHB is given. Specially, the introduction is divided in three topics: (1) the basic elements in AHB system and the function of each one, (2) what is the interconnection between these elements and the main signals, and (3) how to do a basic transfer.
**Main Elements of AHB System**

The main elements are the master/s, the slave/s, one arbiter, and one decoder. A brief description of each element is shown below [3]:

1) **MASTER:**
   It is able to start read and write operations, providing the address and control signals of the transfer. AHB bus supports a master-slave communications with multiple masters, but only one master can transfer at the same time; in other words, more than one master cannot use the bus simultaneously. Examples of masters are the microprocessor, the errors interface, or Direct Memory Access (DMA).

2) **SLAVE:**
   Its function is to answer the read and write operations in a given address rank. The slave generates specific signals which indicate the state of the transfer to the active master. Examples of slaves are internal memories or external memory interfaces.

3) **ARBITER:**
   Its function is to decide which master can use the bus at each moment. The arbitration algorithm can be implemented in accordance with the application necessities. It is important to point out that every design must have only one arbiter.

4) **DECODER:**
   Its function is to decode the address sent by the active master with the goal to select the slave involved in the transfer. Only one decoder is needed in the same system.

**Bus Interconnection**

Firstly, note that a list of the AHB signals and arbitration signals is shown in Appendix A. In addition, a brief description of each signal is given, helping to understand the design.

The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme, as can be seen in Figure 2. With this interconnection, all masters can generate the address and control signals of the transfer that they want to do, and the arbiter decides which signals (from the same master) arrive to the slaves. On the other side, a decoder is needed to select the slave whom the active master is communicating.
BASIC TRANSFER

Before a transfer between a master and a slave starts, the master must request the bus to the arbiter using a requesting signal, HBUSREQx. After that, the arbiter can give access to the bus asserting HGRANTx signal of the requesting master. Later, when HREADY is asserted, the transfer can start.

It is important to highlight that AHB transfer consists of an address and control cycle, and one or more cycles for the data. Therefore, the master starts a transfer by setting up the address and control signals. These signals indicate the transfer direction (read or write), data width, and if the transfer belongs to a burst. This information is available only for one clock cycle, hence all the slaves have to read this information at this time.

Two buses are used to transfer data: one for write operations, and one for read operations. The write data bus is used to transfer from masters to slaves, while read...
data bus is used in the other direction. Unlike control signals, data can be in the bus for the time that is needed; just keep HREADY low is necessary.

During the transfer, the slave indicates the state of the transfer with HRESP signal. This signal can take four values: OKAY when the transfer is done normally; ERROR when something wrong has happened; RETRY and SPLIT when the transfer has to be interrupted and resumed later [3].

At this point, an idea about how to do a transfer is had. However, a simple transfer example might clarify the procedure: one clock cycle for control signals and address, and one clock cycle for data (see Figure 3). In this transfer, the master writes the address and control signal before the first rising edge clock; the slave saves these values in the next edge. Afterwards, the slave sends its answer which is received by the master in the third edge.

![Figure 3: Simple transfer [3]](image)

The slave can extend data phase keeping HREADY low, as can be seen in Figure 4. It allows slave to have additional time to complete the transfer. Regarding to write transfers, master must keep data in the bus during all data phase until HREADY is not asserted. The same happens in read transfers: the slave does not have to write data in the bus until HREADY is not asserted by the slave.
Address phase and data phase occur in different clock cycles. In fact, address phase of one transfer overlaps with data phase of the previous transfer. This overlapping of address and data is fundamental to the pipelined nature of the bus and allows for high performance operation, while still providing adequate time for a slave to provide the response to a transfer. As can be seen in Figure 5, if a data phase of a transfer is extended, address phase of the next transfer will also be extended. Despite this, slaves only read control signals when HREADY is high.

More information about AMBA AHB may be found in [3].
2.2 CAN Bus

CAN bus (Controller Area Network) is a multi-master broadcast serial bus standard for connecting electronic control units, and it was developed by Robert Bosch. Each node is able to send and receive messages, but not at the same time. A message consists primarily of an ID (identifier), which represents the priority of the message, and up to eight data bytes. This information is transmitted serially by the bus.

If the bus is free, any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits - it represents, zeroes -) will overwrite other nodes less dominant IDs, so that only the dominant message will remain and will be received by all nodes. This mechanism is referred to as priority based bus arbitration. Messages with numerically smaller values of IDs have higher priority and are transmitted first.

Some of the basic features of this bus are:

- Message priority
- Formed by only two wires
- Flexible configuration
- Multicast communication
- Multi-master system

Due to this bus only have two wires, information that is sent in the bus has to be encapsulated and transmitted bit to bit. The frame format is illustrated in Figure 6:

![Figure 6: Frame format of CAN bus [4]](image)

The Data Frame begins with a dominant Start of Frame (SOF) bit for hard synchronization of all nodes. The SOF bit is followed by the Arbitration Field reflecting content and priority of the message. The next field is the Control Field which specifies mainly the number of bytes of data contained in the message.
The Cyclic Redundancy Check (CRC) Field is used to detect possible transmission errors. It consists of a 15-bit CRC sequence completed by the recessive CRC delimiter bit. During the Acknowledgement (ACK) Field the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit. The recessive bits of the End of Frame end the Data Frame. Between two frames there must be a recessive 3-bit Intermission field.
3. System approach

This chapter describes the main specifications and goals of this project, evaluates various design alternatives, and finally chooses the better one, using reasoned explanations.

3.1 Specifications

In this section, two kinds of specifications are presented (behavioural and electrical specifications). Behavioural specifications are related to what the system must do and they will be used to implement the VHDL code. On the other hand, electrical specifications will be used to find a simple, but accurate, wire model and the delay of the wire in this system. These specifications are shown below.

**Behavioural Specifications**

- The system under design has to be formed by different modules (N), each one able to send and receive data.
- Every module must be able to send 8 bits (1 byte) of data, at least, to the other modules of the system that have an upper identifier than the sender (see Figure 7).

![Figure 7: System architecture](image)

At the end, second module must have its own data (D1) and data from modules with lower identifier (only D0). On the other hand, last module must have its own data (DN-1) and also data from modules with lower identifier (all data packets; from D0 until DN-2).
- Interconnection between these modules must be done by a shared bus. AMBA bus is recommended, but other options can be analysed.
- The placement of these modules, on the chip, should be designed in order to minimize the delay. The distribution may be on a one-dimensional system, as Figure 7 shows, or on a two-dimensional system.

**ELECTRICAL SPECIFICATIONS**

- To find a wire model, four wires around each one are considered. Figure 8 describes exactly the wires that are important to consider, the distances between them and useful electrical parameters.

![Figure 8: The wire](image)

\[
\begin{align*}
W &= 100\text{nm}, \quad H = 200\text{nm} \\
\text{dist}L &= 100\text{nm}, \quad \text{dist}H = 150\text{nm} \\
k &= 3 \\
R_{\Box} &= 150\text{m}\Omega
\end{align*}
\]

3.2 **Design alternatives**

Before start any design it is very important to analysed different alternatives. This section considers three alternatives and mentions their advantages and their drawbacks. Finally, one of these alternatives is chosen to implement.
**ALTERNATIVE 1**

The first alternative use AMBA bus, and every module is formed by a master and a slave. A scheme of this system, with 8 modules, is illustrated in Figure 9.

In this case, each module sends its own data to other modules with upper identifier. Note that:

- **These transfers happen serially:** first, one module sends its data to all other modules. After that, another module sends data and so on.
- **The order of the transfers is determined by the Arbiter.** It is no necessary that M0 sends data in first place, M1 in second place, etc. Nevertheless, in this design, the arbitration algorithm will give priority to the module with lowest ID.
- **At the end, each module has data packets that are shown below them.**

Finally, Table 1 shows the advantages and the drawbacks of this design alternative.

<table>
<thead>
<tr>
<th>ADVANTAGES</th>
<th>DRAWBACKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA bus has a parallel data bus that allows all bits of the packet are sent at the same time.</td>
<td>Due to AMBA bus is used, all modules must be master and slave at the same time. It denotes that it is necessary N masters and N slaves for a system with N modules. If N is big, total transfers will take a</td>
</tr>
</tbody>
</table>
In a broadcast architecture, the choice of on-chip bus standards is crucial. A prohibitive time, because transfers happen serially. It is necessary an arbiter to grant the bus.

Table 1: Advantages and drawbacks of alternative 1

**ALTERNATIVE 2**

The second alternative uses CAN bus. It denotes that when a module sends its own data, all other modules can read this data at the same time. Therefore, using this alternative, only N transfers are required, even it is important to remember that every transfer is done bit to bit [4] [5]. Hence, some clock cycles are needed for each transfer (see section 2.2 for more information).

The advantages and drawbacks of this alternative are summarized in Table 2.

<table>
<thead>
<tr>
<th>ADVANTAGES</th>
<th>DRAWBACKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>The frame sent by a module, can be read for all other modules at the same time.</td>
<td>The transfer of a frame is done bit to bit because there are only two wires. Consequently, although only N frames are sent, the total transfers will take a prohibitive time.</td>
</tr>
<tr>
<td>Only two wires are required for this bus. Therefore, lower resources than AMBA bus.</td>
<td>Arbiter is not necessary because arbitration process is done at the beginning of the frame, using dominant bits.</td>
</tr>
</tbody>
</table>

Table 2: Advantages and drawbacks of alternative 2

**ALTERNATIVE 3**

After alternatives 1 and 2 have been analysed, consider a third alternative, that combines features of the first and second one, might be a good option. The idea is take the first alternative (with AMBA bus) as a base and perform some modifications. The main idea is not use address bus of AMBA, because when a module is transmitting data, all other modules should read this data at the same time (hence, transfers from the same module do not happen serially). In addition, there will be other differences between original AMBA signals and used signals; these differences are shown in section 4.1.

The advantages and drawbacks of these alternatives are summarized in Table 3.

<table>
<thead>
<tr>
<th>ADVANTAGES</th>
<th>DRAWBACKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA bus has a parallel data bus that allows all bits of the packet are sent at the same time.</td>
<td>An arbiter and control signals are needed. However, not all AMBA control signals will be used.</td>
</tr>
<tr>
<td>In this case, transfers do not happen serially (total transfers time is lower).</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Advantages and drawbacks of alternative 3
3.3 System block diagram

At this point, the alternative that will be implemented is known. Below, the blocks that comprise the system are shown, as well as the interconnection between blocks.

The whole system is illustrated in Figure 11. Note that there is not any decoder because the alternative that is implemented does not use address bus. Therefore, no decoder is needed in the design. Nevertheless, in section 4.1.4, there is a description of the decoder design (for example, decoder might be used in alternative 1). In addition, remember that each module is formed by a master and slave, as can be seen in Figure 10.
Figure 11: Block diagram of the system
4. Design

This chapter shows the design of the system (using VHDL), and its blocks are explained. Moreover, the wire model is calculated and designed: calculation of the wire parameters, calculation of the delay per millimetre and the energy per bit and, finally, the implementation of the wire model using VHDL.

4.1 AMBA AHB system (VHDL)

In this section, the VHDL description is shown. Remember that the system needs four specific blocks (Master, Slave, Arbiter, and Decoder). Due to this, each module is described in the next lines, paying special attention on the main goal, the basic features, and its implementation. In addition, it is important to highlight that, as has been presented in section 3.2, the system is designed using AMBA AHB bus, but not all AHB signals are used in this design. The differences between original signals and used signals are also shown below.

4.1.1 Master

Master is the block that must start a transfer. It has to be able to request for the bus, write the address and control values to the bus, and after that, write or read data to/from the bus.

**Features**

This block has the interface of signals [3] that is shown in Figure 12, and the functions of these signals are explained in Table 4.

![Figure 12: Master AHB block](image-url)
On-chip bus standards in a broadcast architecture

### Signal | Direction (bits) | Description
--- | --- | ---
HCLK | In (1) | Clock of the system. All signal timings are related to the rising edge of HCLK.
HRESETn | In (1) | Reset of the system. Active low.
HBUSREQ | Out (1) | Indicates that the master requires the bus.
HGRANT | In (1) | Indicates that the master is currently the highest priority master in the system and has access to the bus.
HWDATA | Out (8) | The write data bus is used to transfer data from the master to the slaves.
HREADY | In (1) | When HIGH, the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer (by the slave).
HTRANS | Out (2) | Indicates the type of the current transfer. It would be enough use only 1 bit.
HRESP | In (2) | Indicates additional information on the status of the transfer. It would be enough use only 1 bit.

<table>
<thead>
<tr>
<th>Differences between original signals and used signals:</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is no read data bus in this Master block. Remember that the master only needs write information to the slaves.</td>
</tr>
<tr>
<td>There is no address bus because this Master block writes data to all slaves at the same time. Therefore, there is no need to identify the slaves.</td>
</tr>
<tr>
<td>AMBA specification uses other signals that are not used in the design:</td>
</tr>
<tr>
<td>- HSIZE: Indicates the size of the transfer; but all our transfers have a size of 8 bits.</td>
</tr>
<tr>
<td>- HBURST: Indicates if the transfer forms part of a burst. Nevertheless, all our transfers are simple.</td>
</tr>
<tr>
<td>- HPROT: Provides additional information. No necessary for this design.</td>
</tr>
</tbody>
</table>

### Implementation

The procedure that must be followed to make a simple transfer is shown in Figure 3. Before that, the master must request the bus, asserting HBUSREQ signal until HGRANT does not become active high. Afterwards, the current master has access to the bus and may start the simple transfer. The Algorithmic State Machine (ASM) of this block is shown in Figure 13.
It is important to highlight that the function of state S0 is to request the bus. After that, state S1 waits for the response from the arbiter (assertion of HGRANT) and immediately the master indicates that the transfer will be a simple transfer (HTRANS="10"). Afterwards, the master waits in state S2 until all slaves have read control signals (HREADY='1') and then, data is written in the bus and is read by slaves. Finally, if the master has new data to send, the procedure starts again. Otherwise, the master waits in state S3.
4.1.2 Slave

Slave is the block that must wait for a write or read request from one master. In the design, the slave waits for a write request, which is formed by two phases: control phase and data phase.

**Features**

This block has the interface of signals [3] that is shown in Figure 14.

![Figure 14: Slave AHB block](image)

Note that the signals of this block are approximately the same as the Master AHB has. The only new signal is HMASTER, which indicates the ID of the master that is currently the owner of the bus.

- **Differences between original signals and used signals:**
  - All the differences commented in section 4.1.1 apply in this section.
  - In addition, original AMBA protocol uses HSEL signal, whose role is select the slave that the master are writing to. This signal comes originally from the decoder. However, as has been previously mentioned, all slaves read data at the same time, thus this signal is not necessary.

**Implementation**

Figure 15 shows the ASM of the slave block, which is formed by two states (S0 and S1). The first one, S0, waits for HTRANS="10" (it waits until a simple transfer from a master is initiated). After that, the slave asserts HREADY (if it is not busy)
and stores, in signal pos, the ID of the master that is transmitting. This signal will be used to store the received data in the corresponding position of a vector that stores data from all masters. Afterwards, in state S1, data from the master is stored, and a response is given (HRESP="00" – Okay). Thereafter, if there is not any new transfer (HTRANS≠"10"), the state machine goes to state S0. Otherwise, the ID of the new master is stored in signal pos and the procedure of state S1 is repeated.

Figure 15: ASM of Slave AHB block

4.1.3 Arbiter

The roles of this block are to manage the access to the bus and to provide the HMASTER signal (ID of the current master that has access to the bus).

FEATURES

This block has the interface of signals [3] that is shown in Figure 16, and the functions of these signals are explained in Table 5.
On-chip bus standards in a broadcast architecture

Figure 16: Arbiter block

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction (bits)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCLK</td>
<td>In (1)</td>
<td>Clock of the system. All signal timings are related to the rising edge of HCLK.</td>
</tr>
<tr>
<td>HRESETn</td>
<td>In (1)</td>
<td>Reset of the system. Active low.</td>
</tr>
<tr>
<td>HBUSREQ</td>
<td>In (N)</td>
<td>This bus is formed by all HBUSREQ bits that come from the masters.</td>
</tr>
<tr>
<td>HGRANT</td>
<td>Out (N)</td>
<td>This bus is formed by all HGRANT bits that go to the masters.</td>
</tr>
<tr>
<td>HMASTER</td>
<td>Out (M)</td>
<td>Indicates the ID of the master that has access to the bus. The ID is expressed in binary code.</td>
</tr>
</tbody>
</table>

Table 5: Inputs and outputs of Arbiter block

- Differences between original signals and used signals:
  - Original HMASTER signal has four bits (it denotes a maximum of sixteen masters in the system). In this design, M bits are used because more than sixteen masters will be probably necessary.
  - In addition, there are three arbitrations signals that are not used:
    - HLOCKx: When high, this signal indicates that the master requires locked access to the bus. This feature is not needed in the design.
    - HMASTLOCK: Indicates the ID of the master which is performing a locked sequence of transfers.
    - HSPLITx[15:0]: It is used by a slave when it can re-attempt a transfer that has been stopped before, due to the slave was busy. This feature is not implemented in the design.

Implementation

Figure 17 shows the ASM of the arbiter block, which is formed by two states machines. The first one (a) is used to arbitrate the access to the bus, while the second one (b) is used to provide HMASTER signal.
In (a), ‘j’ is a variable of a “for loop”. It means that its changes are immediate and it is not necessary wait until the next rising edge for its update.
The arbitration process starts waiting for the assertion of any bit of HBUSREQ bus. It denotes that there is, at least, one master that wants to access the bus. Immediately, there is a “for loop” looking for the master that requires the bus. When it is found, the corresponding HGRANT signal is asserted. Afterwards, in state S1, the process waits until deassertion of HBUSREQ from the master that has the bus and, posteriorly, HGRANT signal is deasserted. Finally, if there are more masters that are requesting the bus, the process of the state S1 will start again. Otherwise, the state machine will go to state S0.

On the other hand, HMASTER generation process consists on compare the actual master that has the bus with the previous master that had the bus. While these values are equal, no action is required because the owner of the bus have not changed. Otherwise, if the values are different, HMASTER signal will be updated with the ID of the new master.

4.1.4 Decoder

The decoder in an AMBA system is used to perform a centralized address decoding function, which improves the portability of peripherals, by making them independent of the system memory map. In our design, the decoder is not used because address bus is not used [3]. However, this block is implemented and explained because it can be useful in future configurations, where address bus is used.

**Features**

The interface of signals of this block is shown in Figure 18. Note that it is a very simple interface, with one input and one output. The input signal is the address bus, while the output is corresponding to HSEL signal (commented in section 4.1.2). In addition, it is important to point out that this block is totally combinational (without clock signal) because the block cannot introduce any cycle delay.

![Decoder block](image)
IMPLEMENTATION

The implementation of this block is done using a VHDL process without using a clock. This process is shown in Figure 19.

```vhdl
process(SEL)
begin
    for i in OUTPUT'RANGE loop -- SEL range is HADDR[bits-1 downto 0]
        if (conv_integer(SEL)=i) then -- Bin-to-Integer conversion
            OUTPUT(i)<=1;
        else
            OUTPUT(i)<=0;
        end if;
    end loop;
end process;
```

Figure 19: VHDL process of Decoder block

Input SEL is in the sensitivity list of the process. Therefore, every time that SEL changes, this process will run and will decode the input to an output vector of N bits (N is the total number of slaves). For instance, if SEL=“11”, then OUTPUT(3)=’1’.

4.2 Wire model

This section is divided in three parts. The first one shows the calculation of the wire parameters, such as resistance and capacitance. The second one shows the delay of the wire (with and without repeaters) and the energy consumed per useful bit. Finally, last part shows the implementation of the wire model using VHDL.

4.2.1 Calculation of wire parameters

First, as has previously commented in the specifications section, Figure 8 must be considered to calculate the parameters of the wire, where the main wire is surrounded by four more wires.

To calculate the resistance of the wire (per length), equation (1) is used [6] [7] [8]:

\[ R = \rho \cdot \frac{L}{\text{Area}} = \rho \cdot \frac{L}{H \cdot W} \quad \text{where} \quad \rho = R \cdot \pi \cdot H \]

(1)
Therefore:

\[ R_{\text{wire}} = R \cdot \frac{L}{W} = \frac{150m\Omega}{100nm} \cdot L = 1.5 \cdot L \cdot k\Omega \rightarrow R_{\text{wire}} = 1.5 k\Omega/mm \]  

(2)

**NOTES (APPLY FOR ALL DOCUMENT):**

- L is the length of the wire in millimetres.
- To differentiate between distributed (per unit length) wire parameters versus total lumped values, lowercase will be used to denote the former and uppercase for the latter.

Next parameter to find is the capacitance of the wire. Capacitance might be modelled by four parallel-plate capacitors for the top, bottom, right, and left sides [9]. In addition, as can be seen in Figure 20, total capacitance is formed by the contribution of various capacitances: inter-wire, ground, and parallel-plate capacitances [6]. However, according to the specifications values (W/H=0.5), the total capacitance might be approximated by inter-wire (or wire-to-wire) capacitance. Therefore, other contributions will be disregarded and the scenario of Figure 21 will be considered.

![Figure 20: Effect of wire capacitances [6]](image)

When \( W \) becomes smaller than 1.75 \( H \), the inter-wire capacitance starts to dominate (red line).

**Our case: W/H=0.5** (green line)

\[ C_{\text{total}} \approx C_{\text{interwire}} \]
To calculate the capacitances, parallel-plate model is used, without considering fringe capacitance [6] [8]:

\[ C = \frac{\varepsilon_{ox} \cdot \text{Area}}{\text{distance}}, \text{where } \varepsilon_{ox} = \varepsilon_r \cdot \varepsilon_0 = k \cdot \varepsilon_0 = 3 \cdot (8.854 \cdot 10^{-12}) = 26.56 \text{ pF/m} \]

\[ C_h = \frac{\varepsilon_{ox} \cdot H \cdot L}{\text{distL}} = \frac{\varepsilon_{ox} \cdot 200\text{nm} \cdot L}{100\text{nm}} = 53.12 \cdot L \text{ fF} \rightarrow c_h = 53.12 \text{ fF/mm} \]  

(3)

\[ C_v = \frac{\varepsilon_{ox} \cdot W \cdot L}{\text{distH}} = \frac{\varepsilon_{ox} \cdot 100\text{nm} \cdot L}{150\text{nm}} = 17.71 \cdot L \text{ fF} \rightarrow c_v = 17.71 \text{ fF/mm} \]

Therefore, the total wire capacitance is the sum of the four capacitances in Figure 21 [7] [10]:

\[ C_{wire} = 2C_h + 2C_v = 141.66 \cdot L \text{ fF} \rightarrow c_{wire} = 141.66 \text{ fF/mm} \]  

(4)

4.2.2 Wire delay and energy per bit

In this subsection, wire delay (per millimetre) and energy per useful bit are found. These parameters will be useful to characterize the system and know its viability.
**Wire delay**

At this point, it is necessary to find the wire model to calculate the delay \( T_{RC} \), which will be a function of the length \( L \). The simplest one is RC lumped model [6], shown in Figure 22:

![Figure 22: RC lumped model [6]](image)

The Elmore delay of this chain network is shown in equation (5):

\[
T_{RC} = R_1 \cdot C_1 + (R_1 + R_2) \cdot C_2 + \cdots + (R_1 + \cdots + R_N) \cdot C_N = \sum_{i=1}^{N} C_i \sum_{j=1}^{i} R_j \tag{5}
\]

This model can be used as an approximation of a resistive-capacitive wire. Imagine this wire has a total length of \( L \) and it is partitioned into \( N \) identical segments, each with a length of \( L/N \). The resistance and capacitance of each segment are hence given by \( rL/N \) and \( cL/N \), respectively. Using the Elmore formula [6], an approximation of the time-delay of the wire can be obtained:

\[
T_{RC} = \left( \frac{L}{N} \right)^2 (rc + 2rc + \cdots + Nrc) = (rcL^2) \cdot \frac{N(N + 1)}{2N^2} = R \cdot C \cdot \frac{N + 1}{2N} \tag{6}
\]

Note that \( R = r \cdot L \) and \( C = c \cdot L \), and they are the total resistance and capacitance of the wire of length \( L \). In addition, note that for large values of \( N \), expression (7) is obtained:

\[
T_{RC} = R \cdot C \cdot \frac{L^2}{2} \tag{7}
\]

It is important to point out that the delay of a wire is a **quadratic function of its length**. This denotes that doubling the length of the wire quadruples its delay [6].

Following with the calculation, substituting the \( r \) and \( c \) values from (2) and (4) respectively, in equation (7), time-delay \( T_{RC} \) may be found:

\[
T_{RC} = \frac{1}{2} \cdot \frac{1.5k\Omega}{mm} \cdot \frac{141.66fF}{mm} \cdot L^2 = 105.75 \cdot L^2 \text{ ps}, \text{ where } L \text{ is in millimeters.} \tag{8}
\]
The next step is to optimize the delay of a 1mm wire with repeaters insertion. Note that without repeaters and without the effect of the load capacitance, the total delay of this wire is 105.75 ps (8).

However, the load capacitance at the end of the wire must be considered, as Figure 23 shows. This capacitance is given by [7]:

\[ C_{load} = C_{gate} + C_{drain} + C_{body} = \]
\[ 9 \cdot 10^{-16}F + 2.97 \cdot 10^{-16}F + 1.8 \cdot 10^{-16}F = 1.377fF \] (9)

Considering that the \( R_{driver} \) of the circuit is 50Ω, the delay in (10) is obtained (without repeaters) [7]:

\[ D_{total}(without\ repeaters) = (R_{driver} + R_{wire}) \cdot (C_{wire} + C_{load}) - \frac{1}{2} \cdot R_{wire} \cdot C_{wire} \]

\[ D_{total}(without\ repeaters) = 115.46\ ps \] (10)

After this calculation, repeaters are inserted in the design, as Figure 24 shows. Specifically, N-1 repeaters are inserted in a wire formed by N parts.

\[ D_{total}(with\ repeaters) = 115.46\ ps \]
As a result, the total delay is:

\[
D_{\text{total}} = \left( R_{\text{driver}} + r_{\text{wire}} \cdot \frac{L}{N} \right) \cdot c_{\text{wire}} \cdot \frac{L}{N} - \frac{1}{2} \cdot r_{\text{wire}} \cdot c_{\text{wire}} \cdot \left( \frac{L}{N} \right)^2 + D_{\text{repeater}}
+ (N - 2) \cdot \left( \frac{1}{2} \cdot r_{\text{wire}} \cdot c_{\text{wire}} \cdot \left( \frac{L}{N} \right)^2 + D_{\text{repeater}} \right) + r_{\text{wire}}
\]

\[
\cdot \frac{L}{N} \left( c_{\text{wire}} \cdot \frac{L}{N} + C_{\text{load}} \right) - \frac{1}{2} \cdot r_{\text{wire}} \cdot c_{\text{wire}} \cdot \left( \frac{L}{N} \right)^2
\]

To optimize delay, equation (12) must be solved.

\[
\frac{\partial D_{\text{total}}}{\partial N} = 0
\]  

However, to simplify the calculation, the optimization will be done without the effect of \( R_{\text{driver}} \) and \( C_{\text{load}} \). In that case, the delay is shown in equation (13):

\[
D_{\text{total}} = N \cdot 0.5 \cdot r \cdot c \cdot \left( \frac{L}{N} \right)^2 + (N - 1) \cdot D_{\text{repeater}}
\]

As a consequence of the derivation, expression (14) is obtained:

\[
\frac{\partial D_{\text{total}}}{\partial N} = 0 \rightarrow \quad N_{\text{opt}} = L \cdot \frac{0.5 \cdot r \cdot c}{D_{\text{repeater}}} = \frac{D_{\text{wire (unbuffered)}}}{D_{\text{repeater}}}
\]

Subsequently, and considering that the delay of the repeater is about 15ps [11], the optimal number of repeaters is obtained in (15):

\[
N_{\text{opt}} = \frac{\sqrt{115.46ps}}{15ps} = 2.77 \rightarrow 3 \text{ sections of wire (2 repeaters)}
\]

Consequently, substituting (15) into (13) gives the optimal delay in a 1mm wire:

\[
D_{\text{total}} = 14.166ps + 15ps + 11.805ps + 15ps + 12.494ps = 68.465ps
\]

Therefore, an improvement of 40.7% in delay is obtained.

**ENERGY PER USEFUL BIT**

Finally, the energy per useful bit is calculated. In this part, would be very interesting to compare the energy per bit in three different scenarios: in a full AMBA system, in the system designed (modified AMBA), and in a point-to-point communication. The calculations are performed below:
**Full AMBA system**

In this case, all control and arbitration signals of AMBA AHB system are considered. Therefore, every module of the system (master and slave) would have [3]:

- 64 bits of data (32 bits of write data and 32 bits of read data)
- 17 bits of control and 32 bits of address
- 3.3125 bits of arbitration

It represents that there are 64 useful bits \( (u\_bits) \) per 132.3125 total bits (per each module). Consequently, the energy per bit [12] without considering the energy of the buffers \( (E_b') \) is given in (17):

\[
E_b' = C_{wire} \cdot V^2 \cdot \frac{num\_bits}{num\_u\_bits} = 141.66 \cdot \frac{fF}{mm} \cdot (1.2V)^2 \cdot \frac{132.3125 \text{ bits}}{64 \text{ u\_bits}} = \frac{0.422 \text{ pJ}}{\text{bit}} \quad (17)
\]

On the other hand, every buffer consumes energy at switch time. Considering that a buffer is a CMOS inverter (it denotes that \( C_{in}=2 \cdot C_{gate} \)), the energy that consumes each buffer per useful bit is:

\[
E_{buff} = C_L \cdot V^2 \cdot \frac{num\_bits}{num\_u\_bits} = \left( C_{wire} \cdot l_{buff} + 2 \cdot C_{gate} \right) \cdot (1.2V)^2 \cdot \frac{132.3125 \text{ bits}}{64 \text{ u\_bits}}
\]

\[
= \left( 141.66 \cdot \frac{fF}{mm} \cdot 0.33mm + 2 \cdot 0.9fF \right) \cdot (1.2V)^2 \cdot \frac{132.3125 \text{ bits}}{64 \text{ u\_bits}}
\]

\[
= 0.145 \text{ pJ} \text{ bit}^{-1}
\]

Finally, using equations (17) and (18), it is possible to calculate the total energy per useful bit (remember that there are 2 buffers in a length of 1mm).

\[
E_b = E_b' + \frac{2\text{buff}}{mm} \cdot E_{buff} = 0.711 \frac{\text{pJ}}{\text{bit} \cdot \text{mm}}
\]

**Modified AMBA system**

In this case, only the signals described in section 4.1 are considered to calculate the \( E_b \). Therefore, every module has:

- 8 bits of data
- 3 bits of control
- 2 bits of arbitration

It represents that there are 8 useful bits per 13 total bits. Therefore, \( E_b' \) and \( E_{buff} \) are given by equations (20) and (21).
Finally, using equations (20) and (21), the total $E_b$ is obtained.

$$E_b = E'_b + \frac{2\text{buff}}{\text{mm}} \cdot E_{\text{buff}} = (0.331 + 0.227) \frac{\text{pj}}{\text{bit} \cdot \text{mm}} = 0.558 \frac{\text{pj}}{\text{bit} \cdot \text{mm}}$$  \tag{22}$$

\section*{Point-to-point communication system}

In the last case, the arbitration signals are not considered because every link has its own wire. In addition, there are no control signals because a point-to-point system can be operated synchronously. Therefore, every module would have 8 bits of data and hence, 8 useful bits. Consequently, $E'_b$ and $E_{\text{buff}}$ are:

$$E'_b = 141.66 \frac{fF}{\text{mm}} \cdot (1.2V)^2 = 0.204 \frac{\text{pj}}{\text{bit} \cdot \text{mm}}$$  \tag{23}$$

$$E_{\text{buff}} = \left(141.66 \frac{fF}{\text{mm}} \cdot 0.33\text{mm} + 2 \cdot 0.9fF\right) \cdot (1.2V)^2 = 0.070 \frac{\text{pj}}{\text{bit} \cdot \text{mm}}$$  \tag{24}$$

Finally, using (23) and (24), the total $E_b$ is obtained:

$$E_b = E'_b + \frac{2\text{buff}}{\text{mm}} \cdot E_{\text{buff}} = (0.204 + 0.140) \frac{\text{pj}}{\text{bit} \cdot \text{mm}} = 0.344 \frac{\text{pj}}{\text{bit} \cdot \text{mm}}$$  \tag{25}$$

\subsection*{4.2.3 Wire delay model in VHDL}

Once delay per millimetre is known, it is necessary decide the placement of the modules. After that, the distance between modules will be known, and VHDL model may be implemented.

The placement proposed in this project is shown in Figure 25. Note that the modules are distributed on a two-dimensional system because the distances between the modules are shorter than on a one-dimensional system. Consequently, the delay is also smaller.

Remember that any module that wants to transfer must request the bus to the arbiter. In Figure 25, this request costs three steps of 0.316mm (worst case –
request from orange modules). Consequently, the answer from the arbiter to the module costs three steps as well.

On the other hand, when a module from the last region (orange modules in Figure 25) sends data to other module in the last region, the distance is six steps. Therefore, it is the worst case to calculate over how many modules the system can transfer.

Note that the distance between two modules is a function of their region ID. Considering that the modules with the same colour (in Figure 25) belong to the same region, and regions are formed by the modules shown in Table 6, the distance between two regions (Rx and Ry) is \(x+y \text{ steps of 0.316 mm}\).

<table>
<thead>
<tr>
<th>Region</th>
<th>Formed by modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>0 to 3</td>
</tr>
<tr>
<td>R2</td>
<td>4 to 11</td>
</tr>
<tr>
<td>R3</td>
<td>12 to 23</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Rn</td>
<td>2(n-1)n to 2n(n+1)-1</td>
</tr>
</tbody>
</table>

Table 6: Regions and modules

At this point, it is important to know the maximum distance between two modules. Obviously, it depends on the total area of the system. Therefore, it is
necessary to show the maximum distance as a function of the area. First, note
that every side of the square should be formed by an odd number of elements (0.1
mm$^2$ each). In this case, the configuration is totally symmetric respect the centre
element (arbiter and multiplexers) and the calculation is easier.

Consequently, two maximum distances might be calculated:

1) Considering only the modules of the complete regions (coloured modules in
Figure 25):

\[
Max\_distance_1 = \left( \left\lfloor \sqrt{\frac{\text{Area}(mm^2)}{0.1 \text{ mm}^2}} \right\rfloor - 1 \right) \cdot 0.316\text{mm}
\] (26)

2) Considering all modules in the square:

\[
Max\_distance_2 = 2 \cdot \left( \left\lfloor \sqrt{\frac{\text{Area}(mm^2)}{0.1 \text{ mm}^2}} \right\rfloor - 1 \right) \cdot 0.316\text{mm}
\] (27)

**NOTE:** In equations (26) and (27), floor function is used. It returns the previous
integer of its result.

Therefore, when total area is known, maximum distance may be found.
Consequently, with maximum distance, it is possible to find over how many
regions the system might transfer and also, using Table 6, over how many
modules.

Finally, the last step is to implement the delay model in VHDL. To carry it out, it
is necessary a function that returns the region ID depending on the module ID
(Table 6). Hence, all distances in our configuration will be known.

In addition, every module must have delay blocks for all input and output
signals. It is important to highlight that these blocks introduce a delay
corresponding to the distance between the module and the centre of configuration
(arbiter and multiplexers). Hence, all paths become modelled.

Figure 26 shows an example of how the delay of arbitration signals is carried
out (in Master AHB). Note that:

- The signals that end with “_aux” are the input and output signals to the master
  block. Remember that HBUSREQ is an output signal, and HGRANT is an
  input signal. Thus, HBUSREQ_aux is the HBUSREQ signal (from the arbiter)
On-chip bus standards in a broadcast architecture

- Delays are implemented using “after statements” inside a process. The sensitivity list of this process is formed by the signal that must be delayed. Thus, the new signal is updated “reg*STEP_DELAY” after the non-delayed signal has changed.

- “reg*STEP_DELAY” is a time constant: reg indicates the region of the module (besides the steps between the module and the center), and STEP_DELAY indicates the delay per each step. In this case, the step is 0.316mm and its delay is 21.651ns.

```
175  process(HBUSREQ_aux)
176     begin
177     HRBUSREQ<=HBUSREQ_aux after reg*STEP_DELAY;
178     end process;
179
180  process(HGRANT)
181     begin
182     HGRANT_aux<=HGRANT after reg*STEP_DELAY;
183     end process;
```

Figure 26: VHDL delay model: after statements
5. Verification

Finally, it is important to verify the behaviour of all design. In this chapter, two simulations of an 8 modules system are shown. It is significant to note that both simulations are without the delay model due to simulations with delay don't add information. This is because the clock frequency is 10MHz (period=100ns) and delays are insignificant compared to the period.

**SIMULATION 1:**

Simulation 1 shows all AMBA signals in an 8 modules system. Hence, how all modules request the bus and send data to other modules may be validated. Next points give a complete description of the procedure:

1) After reset, all modules request the bus using HBUSREQ signal.
2) The arbiter decides which module will have the bus. In this case, the arbiter asserts HGRANT(0), thus M0 is the new owner of the bus. Note that in the next rise edge, HBUSREQ(0) is deactivated.
3) Once M0 has the bus, next step is to write the control signals. Note that HTRANS="10" (indicates that the transfer is starting).
4) Next cycle, data must be written in the bus. Note that data from M0 is "00000001". In addition, it is important to highlight that this cycle is the last cycle that master needs the bus. Therefore, the arbiter asserts HGRANT(1) to give access to M1 in the next cycle (as long as HREADY is asserted).
5) Finally, in the next cycle, HREADY is asserted and HMASTER signal changes to "001" (indicates that the new owner is M1). After that, the procedure starts again.

**SIMULATION 2:**

Simulation 2 shows all signals of a module, in particular module 4 (M4).

1) First, M0 has the bus (HMASTER="000"). Note that HTRANS_S (S=slave) changes to "10", thus the transfer of M0 starts.
2) Then, M0 writes its data to the bus and this data arrives to all other modules.
3) In this rise edge, data from M0 is stored in “data_vector” signal, whose function is to store data from all modules in the system. Note that points (1), (2) and (3) are repeated until M4 gets the bus.
4) This is the first rise edge that both HGRANT(4) and HREADY are asserted, thus the transfer of M4 starts, following the same steps described above.
5) Once its transfer is finished, M4 continues receiving and storing data from other modules.
On-chip bus standards in a broadcast architecture
6. Conclusions

Once all design has been implemented and verified, the main goals have been achieved. It denotes that a broadcast communication using a shared bus (AMBA) has been implemented. In addition, a wire delay model has been designed and implemented using VHDL, and energy per bit has been calculated. However, concerning to this, it is important to highlight some points:

1) Full AMBA system is not used. The design is performed using only the signals that are needed. Consequently, the performed design uses 84.6% bits less than a full AMBA system.

2) The delay of the wire is a quadratic function of its lengths. This represents that doubling the length of the wire quadruples its delay.

3) Using a buffering architecture, an improvement of 40.7% in delay is obtained.

4) Delay model in VHDL has been implemented, even though it has not provided additional information due to:
   a. Transfers are very quick because the buffering architecture has been optimized for speed. It represents, for instance, that a system of 800 modules has a maximum length of 12mm and, therefore, a maximum delay of 1ns.
   b. The clock frequency is 10MHz, thus the clock period is 100ns. It denotes that a delay of 1ns is only 1% of the clock period.

5) The performed design has an energy per bit of 0.558 \( \frac{pj}{bits\cdot mm} \). It is:
   a. 21.52% lower than \( E_b \) in a full AMBA system (0.711 \( \frac{pj}{bits\cdot mm} \)). Therefore, it is a good decision not to use full AMBA and use a modified AMBA, using only the signals that are useful for the design.
   b. 62.21% higher than \( E_b \) in a point-to-point communication system (0.344 \( \frac{pj}{bits\cdot mm} \)). However, in this case, it is important to point out that a point-to-point communication may not be a good option because many wires are needed in a system with a huge number of modules.

6) The modules are distributed on a two-dimensional system because the distances between the modules are shorter than on a one-dimensional system. Consequently, the delay is also smaller.
Finally, it is important to highlight that this design has been optimized in speed, but not in energy. Next step would be the optimization in energy and, therefore, reduce the number of buffers. Obviously, it will have a negative impact on speed, but the goal is to get a **compromise between energy and speed**. In addition, this kind of optimization will reduce the energy differences between the performed design and a point-to-point communication system.
On-chip bus standards in a broadcast architecture

7. References


[7] KNEPPER, Ronald W. Slides “Introduction to CMOS logic circuits”. Boston University. URL: people.bu.edu/rknepper/sc571/chapter4_a.ppt


Appendix A. AHB signals

2.2 AMBA AHB signal list

This section contains an overview of the AMBA AHB signals (see Table 2-1). A full description of each of the signals can be found in later sections of this document.

All signals are prefixed with the letter H, ensuring that the AHB signals are differentiated from other similarly named signals in a system design.

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCLK</td>
<td>Clock source</td>
<td>This clock times all bus transfers. All signal timings are related to the rising edge of HCLK.</td>
</tr>
<tr>
<td>Bus clock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HRESETn</td>
<td>Reset controller</td>
<td>The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.</td>
</tr>
<tr>
<td>HADDR[31:0]</td>
<td>Master</td>
<td>The 32-bit system address bus.</td>
</tr>
<tr>
<td>Address bus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTRANS[1:0]</td>
<td>Master</td>
<td>Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.</td>
</tr>
<tr>
<td>Transfer type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWRITE</td>
<td>Master</td>
<td>When HIGH this signal indicates a write transfer and when LOW a read transfer.</td>
</tr>
<tr>
<td>Transfer direction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HSIZE[2:0]</td>
<td>Master</td>
<td>Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.</td>
</tr>
<tr>
<td>Transfer size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBURST[2:0]</td>
<td>Master</td>
<td>Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.</td>
</tr>
<tr>
<td>Burst type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HPROT[3:0]</td>
<td>Master</td>
<td>The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.</td>
</tr>
<tr>
<td>Protection control</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1 AMBA AHB signals
### AMBA Signals

Table 2-1 AMBA AHB signals (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWDATA[31:0]</td>
<td>Master</td>
<td>The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.</td>
</tr>
<tr>
<td>HSELx</td>
<td>Decoder</td>
<td>Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.</td>
</tr>
<tr>
<td>HRDATA[31:0]</td>
<td>Slave</td>
<td>The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.</td>
</tr>
<tr>
<td>HREADY</td>
<td>Slave</td>
<td>When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require HREADY as both an input and an output signal.</td>
</tr>
<tr>
<td>HRESP[1:0]</td>
<td>Slave</td>
<td>The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.</td>
</tr>
</tbody>
</table>
AMBA AHB also has a number of signals required to support multiple bus master operation (see Table 2-2). Many of these arbitration signals are dedicated point to point links and in Table 2-2 the suffix x indicates the signal is from module X. For example there will be a number of HBUSREQx signals in a system, such as HBUSREQarm, HBUSREQdma and HBUSREQtlic.

**Table 2-2 Arbitration signals**

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBUSREQx Bus request</td>
<td>Master</td>
<td>A signal from bus master x to the bus arbiter which indicates that the bus master requires the bus. There is an HBUSREQx signal for each bus master in the system, up to a maximum of 16 bus masters.</td>
</tr>
<tr>
<td>HLOCKx Locked transfers</td>
<td>Master</td>
<td>When HIGH this signal indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW.</td>
</tr>
<tr>
<td>HGRANTx Bus grant</td>
<td>Arbiter</td>
<td>This signal indicates that bus master x is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when HREADY is HIGH, so a master gets access to the bus when both HREADY and HGRANTx are HIGH.</td>
</tr>
<tr>
<td>HMASTER[3:0] Master number</td>
<td>Arbiter</td>
<td>These signals from the arbiter indicate which bus master is currently performing a transfer and is used by the slaves which support SPLIT transfers to determine which master is attempting an access. The timing of HMASTER is aligned with the timing of the address and control signals.</td>
</tr>
<tr>
<td>HMASTLOCK Locked sequence</td>
<td>Arbiter</td>
<td>Indicates that the current master is performing a locked sequence of transfers. This signal has the same timing as the HMASTER signal.</td>
</tr>
<tr>
<td>HSPLITx[15:0] Split completion request</td>
<td>Slave (SPLIT-capable)</td>
<td>This 16-bit split bus is used by a slave to indicate to the arbiter which bus masters should be allowed to re-attempt a split transaction. Each bit of this split bus corresponds to a single bus master.</td>
</tr>
</tbody>
</table>
Appendix B. VHDL Code

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.MyDataTypes.all;
use IEEE.NUMERIC_STD.ALL;

entity TOP_AMBA is
END PORT;

HCLK : IN STD_LOGIC;
HRSETn : IN STD_LOGIC
);

end TOP_AMBA;

architecture Behavioral of TOP_AMBA is

COMPONENT CoreAHB
GENERIC (
  numCore : integer range 0 to Ncores-1:=0
);
END COMPONENT;

HCLK : IN std_logic;
HRSETn : IN std_logic;
READY     : IN std_logic;
HRESP_M : IN std_logic;
HGRANT   : IN std_logic;
HTRANS_S : IN std_logic_vector(1 downto 0);
HWDATA_S : IN std_logic_vector(DataWidth-1 downto 0);
HMMASTER : IN std_logic_vector(HADDRbits-1 downto 0);
HTRANS_M : OUT std_logic_vector(1 downto 0);
HWDATA_M : OUT std_logic_vector(DataWidth-1 downto 0);
HUSEQ    : OUT std_logic;
READY    : OUT std_logic;
HRESP_S  : OUT std_logic_vector(1 downto 0)
);

END COMPONENT;

COMPONENT arbiterAHB
PORT(
  HCLK    : IN std_logic;
  HRSETn  : IN std_logic;
  READY   : IN STD_LOGIC;
  HBUSREQ : IN std_logic_vector(Ncores-1 downto 0);
  HGRANT  : OUT std_logic_vector(Ncores-1 downto 0);
  HMMASTER : OUT std_logic_vector(HADDRbits-1 downto 0)
);
On-chip bus standards in a broadcast architecture

```vhdl
TOP_AMBA.vhd

58  }
59  END COMPONENT;
60
61  COMPONENT decoderAHR
62  FORE
63      SEL : IN std_logic_vector(HADDRbits-1 downto 0);
64      OUTPUT : OUT std_logic_vector(Ncores-1 downto 0)
65  )
66  END COMPONENT;
67
68  -- Internal signals --
69  signal HTRANS : STD_LOGIC_VECTOR(1 DOWNTO 0);
70  signal HTRANS_VECT : twobit_vector(Ncores-1 DOWNTO 0);
71  signal HWDATA : STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);
72  signal HWDATA_VECT : data_vector(Ncores-1 DOWNTO 0);
73  signal HREADY : STD_LOGIC;
74  signal HREADY_VECT : STD_LOGIC_VECTOR(Ncores-1 DOWNTO 0);
75  signal HRESP : STD_LOGIC_VECTOR(1 DOWNTO 0);
76  signal HRESP_VECT : twobit_vector(Ncores-1 DOWNTO 0);
77  ------------------------------------------
78  begin
79  ------------------------------------------
80  -- Arbiter signals --
81  signal HBUSREQ : STD_LOGIC_VECTOR(Ncores-1 DOWNTO 0);
82  signal HGRANT : STD_LOGIC_VECTOR(Ncores-1 DOWNTO 0);
83  signal HMASTER : STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0);
84  signal HMASTER_delayed : STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0);
85  ------------------------------------------
86  begin
87  ------------------------------------------
88  -- HREADY DECODER --
89  HREADY <= '1' when not(HREADY_VECT)=0 ELSE '0';
90  ------------------------------------------
91  begin
92  ------------------------------------------
93  -- HWDATA DECODER --
94  HMASTER_delay: process(HCLK)
95  begin
96       if(HCLK='1' and HCLK'event) then
97           HMASTER_delayed<=HMASTER;
98       end if;
99     end process;
100  ------------------------------------------
101  HWDATA <= HWDATA_VECT(conv_integer(HMASTER_delayed));
102  ------------------------------------------
103  begin
104  -- HRESP DECODER --
105     HRESP<=>"00";
106  -- Two bits are used, but it would be enough with 1 bit to indicate "OKAY" and "ERROR".
107  -- In this project, only "OKAY" value is assigned("00"). "ERROR" value is due to physical failure of the bus.
108  ------------------------------------------
109  begin
110  -- HTRANS DECODER --
111     HTRANS <= HTRANS_VECT(conv_integer(HMASTER));
112  ------------------------------------------
```
On-chip bus standards in a broadcast architecture

```
TOP_AMBA.vhd

Sat May 12 20:20:39 2012

113  ------------------------------
114  n CORE_INSTANTIATIONS
115  ------------------------------
116
117
118  Cores: for n in 0 to Nores-1 generate
119    Core_n: CoreAHB
120    GENERIC MAP |
121    numCore => n
122  )
123  PORT MAP(
124    HCLK => HCLK,
125    HRESETn => HRESETn,
126    HTRANS_M => HTRANS_VECTOR(n),
127    HNDDATA_M => HNDDATA_VECTOR(n),
128    HREADY_M => HREADY,
129    HRESP_M => HRESP,
130    HBUSREQ => HBUSREQ(n),
131    HGRANT => HGRANT(n),
132    HTRANS_S => HTRANS,
133    HNDDATA_S => HNDDATA,
134    HREADY_S => HREADY_VECTOR(n),
135    HRESP_S => HRESP_VECTOR(n),
136    HMMASTER => HMMASTER
137 );
138 end generate;
139
140
141  arbiterAHB_0: arbiterAHB PORT MAP{
142    HCLK => HCLK,
143    HRESETn => HRESETn,
144    HREADY => HREADY,
145    HBUSREQ => HBUSREQ,
146    HGRANT => HGRANT,
147    HMMASTER => HMMASTER
148  );
149
150 end Behavioral;
151
```
On-chip bus standards in a broadcast architecture

CoreAHB.vhd

--- Company: Aalto University - Electrical Engineering School
--- Engineer: Romaguera Restudis, Josep-Oriol
--- Design Name: On-Chip bus in a broadcast architecture
--- Module Name: CoreAHB - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.MyDataTypes.all;
use IEEE.NUMERIC_STD.ALL;

class entity CoreAHB is

GElric (  
  numCore : integer range 0 to Ncores-1:=0
);

PORT(
  HCLK : IN STD_LOGIC;
  HRESETn : IN STD_LOGIC;

  -- Master signals --
  HTRANS_M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);  
  HWDATA_M : OUT STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);  
  HREADY_M : IN STD_LOGIC;
  HRESP_M : IN STD_LOGIC_VECTOR(1 DOWNTO 0);

  -- ARBITRATION SIGNALS --
  HBUSEQ : OUT STD_LOGIC;
  HGRANT : IN STD_LOGIC;

  ------------------------

  -- Slave signals --
  HTRANS_S : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
  HWDATA_S : IN STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);
  HREADY_S : OUT STD_LOGIC;
  HRESP_S : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
  HMASTER : OUT STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0)
)

end CoreAHB;

architecture Behavioral of CoreAHB is

COMPONENT masterAHB

GElric (  
  numCore : integer range 0 to Ncores-1:=0
);

PORT(
  HCLK : IN std_logic;
  HRESETn : IN std_logic;
  HREADY : IN std_logic;
  HRESP : IN std_logic_vector(1 downto 0);

Page 1
On-chip bus standards in a broadcast architecture

CoreAHB.vhd

58    HGRANT    : IN std_logic;
59    HTRANS    : OUT std_logic_vector(1 downto 0);
60    HWDATA    : OUT std_logic_vector(NumDataBits-1 downto 0);
61    HBUSREQ    : OUT std_logic;
62    );
63    END COMPONENT;
64
65    COMPONENT slaveAHB
66    GENERIC {
67        numCore    : integer range 0 to Ncores-1:=0
68    },
69    PORT{
70        HCLK    : IN std_logic;
71        HRESETp : IN std_logic;
72        HTRANS  : IN std_logic_vector(1 downto 0);
73        HWDATA  : IN std_logic_vector(NumDataBits-1 downto 0);
74        HMASTER : IN std_logic_vector(HADDRbits-1 downto 0);
75        HREADY  : OUT std_logic;
76        HBUSP   : OUT std_logic_vector(1 downto 0)
77    },
78    END COMPONENT;
79
80    --- INTERNAL SIGNALS ---
81    SIGNAL HTRANS_M_aux  : STD_LOGIC_VECTOR(1 DOWNTO 0);
82    SIGNAL HWDATA_M_aux   : STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);
83    SIGNAL HREADY_M_aux   : STD_LOGIC;
84    SIGNAL HBUSP_M_aux    : STD_LOGIC_VECTOR(1 DOWNTO 0);
85    --- ARBITRATION SIGNALS ---
86    SIGNAL HBUSREQ_aux    : STD_LOGIC;
87    SIGNAL HGRANT_aux     : STD_LOGIC;
88    ------------------------
89
90    --- Slave signals ---
91    SIGNAL HTRANS_S_aux   : STD_LOGIC_VECTOR(1 DOWNTO 0);
92    SIGNAL HWDATA_S_aux    : STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);
93    SIGNAL HREADY_S_aux    : STD_LOGIC;
94    SIGNAL HBUSP_S_aux     : STD_LOGIC_VECTOR(1 DOWNTO 0);
95    SIGNAL HMASTER_aux     : STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0);
96    ------------------------
97
98    signal reg    : integer range 1 to MaxRange;
99
100   begin
101   begin
102   Master: masterAHB
103   GENERIC MAP {
104       numCore => numCore
105   };
106   PORT MAP{
107       HCLK => HCLK,
108       HRESETp => HRESETp,
109       HTRANS => HTRANS_M_aux,
110       HWDATA => HWDATA_M_aux,
111       HREADY => HREADY_M_aux,
112       HBUSP => HBUSP_M_aux,
113       HBUSREQ => HBUSREQ_aux,
On-chip bus standards in a broadcast architecture

CoreAHB.vhd

```
115    HGRANT => HGRANT_aux
116    );
117
118
119   Slave: slaveAHB
120   GENERIC MAP (  
121       numCore => numCore
122   )
123   PORT MAP (  
124       HCLK => HCLK,
125       HRESETn => HRESETn,
126       HTRANS => HTRANS_s_aux,
127       HNDA => HNDA_s_aux,
128       HREADY => HREADY_s_aux,
129       HRESP => HRESP_s_aux,
130       HMASTER => HMASTER_aux
131   );
132
133   -- Calculation of the region ID (depending on the number of the core). "region"  
134   function is described in "MyDataTypes" file.  
135   process(HCLK)  
136     begin  
137       if(HCLK='1' and HCLK'event) then  
138         if (HRESETn='0') then  
139           reg<=region(numCore);  
140         end if;  
141       end if;  
142     end process;  
143
144   -- Delay model is implemented below. This code is only necessary during simulation.  
145   -- All delays are implemented using "after" statements.  
146   ------------------------
147   -- DELAY OF MASTER'S SIGNALS --
148   ------------------------
149   process(HTRANS_M_aux)  
150     begin  
151       HTRANS_M<=HTRANS_M_aux after reg*STEP_DELAY;  
152     end process;
153   process(HWDATA_M_aux)  
154     begin  
155       HWDATA_M<=HWDATA_M_aux after reg*STEP_DELAY;  
156     end process;
157   process(HREADY_M)  
158     begin  
159       HREADY_M_aux<=HREADY_M after reg*STEP_DELAY;  
160     end process;  
161   process(HRESP_M)  
162     begin  
163       HRESP_M_aux<=HRESP_M after reg*STEP_DELAY;  
164     end process;
165   process(HBUSREQ_aux)  
166     begin  
167       HBUSREQ<=HBUSREQ_aux after reg*STEP_DELAY;
```
On-chip bus standards in a broadcast architecture

CoreAHB.vhd

end process;
process(HGRANT)
begin
HGRANT_aux<=HGRANT after reg*STEP_DELAY;
end process;

---------------------
-- DELAY OF SLAVE'S SIGNALS --
---------------------
process(HTRANS_S)
begin
HTRANS_S_aux<=HTRANS_S after reg*STEP_DELAY;
end process;

process(HWDATA_S)
begin
HWDATA_S_aux<=HWDATA_S after reg*STEP_DELAY;
end process;

process(HREADY_S_aux)
begin
HREADY_S<=HREADY_S_aux after reg*STEP_DELAY;
end process;

process(HRESP_S_aux)
begin
HRESP_S<=HRESP_S_aux after reg*STEP_DELAY;
end process;

process(HMASTER)
begin
HMASTER_aux<=HMASTER after reg*STEP_DELAY;
end process;

end Behavioral;
On-chip bus standards in a broadcast architecture

---

masterAHB.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.MyDataTypes.all;
use IEEE.NUMERIC_STD.ALL;

-- Company: Aalto University - Electrical Engineering School
-- Engineer: Romaguera Restudis, Josep-Oriol
-- Design Name: On-Chip bus in a broadcast architecture
-- Module Name: masterAHB - Behavioral

entity masterAHB is
  GENERIC (
    numCore : integer range 0 to Ncores-1:=0
  );
  PORT(
    HCLK : IN STD_LOGIC;
    HRSETn : IN STD_LOGIC;
    HTRANS : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    HWDATA : OUT STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);
    HREADY : IN STD_LOGIC;
    HRESP : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    -- ARBITRATION SIGNALS --
    HBUSREQ : OUT STD_LOGIC;
    HGRANT : IN STD_LOGIC
  );
end masterAHB;

architecture Behavioral of masterAHB is

-- Provisional signals --
signal newData : std_logic:='0';

-- Internal signals --
signal core2transfer : integer range 0 to Ncores;
signal DATA : std_logic_vector(NumDataBits-1 downto 0);

-- State machines --
type state_type is (s0, s1, s2, s3);
signal state : state_type;

begin
  process(HCLK)
  begin
    
Page 1
On-chip bus standards in a broadcast architecture

```vhdl
58     if (HCLK='1' and HCLK'event) then
59         if (HRESETn='0') then
60             -- Reset
61             HTRANS<="00";       -- HTRANS-IDLE --> Nothing to do
62             HBUSREQ<='0';
63             DATA<='MEM(numCore)];       -- Set data of core
64             state<='s0';
65         else
66             case(state) is
67                 when s0 =>
68                     HBUSREQ<='1';
69                     state<='s1';
70                 when s1 =>
71                     if (HGRANT='1' and HREADY='1') then
72                         HTRANS<="10";       -- HTRANS-NONSEQ --> Simple transfer (not burst
73                         transfer)
74                         state<='s2';
75                         HBUSREQ<='0';
76                     end if;
77                 when s2 =>
78                     if (HREADY='1') then
79                         HWDATA<='DATA';
80                         HTRANS<="00";
81                     end if;
82                 when s3 =>
83                     if (newData='1') then
84                         state<='s0';
85                     end if;
86                 when OTHERS =>
87                     state<='s0';
88         end case;
89     end if;
90 end process;
91 end Behavioral;
92
```
On-chip bus standards in a broadcast architecture

slaveA8H.vhd

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.MyDataTypes.all;
6  use IEEE.NUMERIC_STD.ALL;
7
8  entity slaveA8H is
9    GENERIC 
10      numCorec: integer range 0 to Ncores-1:=0
11  );
12
13  PORT(
14    HCLK, : IN STD_LOGIC;
15    HRESETn, : IN STD_LOGIC;
16    HTRANS, : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
17    HBDATA, : IN STD_LOGIC_VECTOR(NumDataBits-1 DOWNTO 0);
18    HREADY, : OUT STD_LOGIC;
19    HRESP, : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
20    HMASTER, : IN STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0)
21  );
22
23  end slaveA8H;
24
25  architecture Behavioral of slaveA8H is
26
27  -- Internal signals --
28  signal DATA_VECTOR: data_vec (Ncores-1 downto 0);
29
30  signal pos: integer range 0 to NCorec:=0;
31  ------------------------
32
33  -- State machines --
34  type state_type is (s0, s1);
35  signal state: state_type;
36  --------------------
37
38  begin
39  process(HCLK)
40  begin
41    if(HCLK='1' and HCLK'event) then
42      if (HRESETn='0') then
43        -- Reset
44        HREADY<='1'; -- Slave are ready
45      end if
46    end if
47  end process;
48
49  \[Aalto University\]
\[School of Electrical Engineering\]
On-chip bus standards in a broadcast architecture

slaveAHB.vhd

---

HRESP<="00"; -- HRESP=OKAY
state<=s0;
DATA_VECTOR(numCore)<=MEM(numCore);
else

  case(state) is

    when s0->
      if (HTRANS="10") then
        -- There is a transfer to receive
        HREADY<="1";
        pos<=conv_integer(HMASTER);
        state<=s1;
        end if;

    when s1->
      -- Receiving: data is stored in "DATA_VECTOR".
      DATA_VECTOR(pos)<=HDATA;
      HRESP<="00";
      HREADY<="1";
      if (HTRANS="10") then
        -- Case of two consecutive transfers to the same slave (not
        common in our design)
        pos<=conv_integer(HMASTER);
        state<=s1;
      else
        state<=s0;
        end if;

    when others->
      state<=s0;

  end case;
end if;
end if;
end process;
end Behavioral;
On-chip bus standards in a broadcast architecture

```vhdl
-- Company: Aalto University - Electrical Engineering School
-- Engineer: Romaguera Restudas, Josep-Oriol
-- Design Name: On-Chip bus in a broadcast architecture
-- Module Name: arbiterAHB - Behavioral

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.MyDataTypes.all;
use IEEE.NUMERIC_STD.All;

entity arbiterAHB is
  PORT(
    HCLK : IN STD_LOGIC;
    HRSETn : IN STD_LOGIC;
    HREADY : IN STD_LOGIC;
    HBUSSIDR : IN STD_LOGIC_VECTOR(NCores-1 DOWNTO 0);
    HGRANT : OUT STD_LOGIC_VECTOR(NCores-1 DOWNTO 0);
    HMASTER : OUT STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0)
  );
end arbiterAHB;

architecture Behavioral of arbiterAHB is

  -- Internal signal --
  signal actual_master , previous_master : integer range 0 to NCores-1;

  -- State machines --
  type state_type is (s0, s1);
  signal state : state_type;

begin

  process(HCLK)
  begin
    if(HCLK='1' and HCLK'event) then
      if (HRSETn='0') then
        -- Reset
        HGRANT<=(others=>'0');
        actual_master<=0;
      else
        case(state) is
          when s0 =>
            if(HBUSSIDR/=0) then
```
arbiterAHB.vhd

58     -- Arbitration algorithm: the lowest ID will have access to the bus.
59     for j in 0 to Ncores-1 loop
60         if(HBUSREQ(j)='1') then
61             HGRANT(j)='1';
62             actual_master<=j;
63             s1<=s1;
64             exit;
65         end if;
66     end loop;
67     else
68         -- If this code executes means that HBUSREQ=0. Then, come back to
69         state0.
70         s1<=s0;
71     end if;
72
73     when s1=>
74         -- When the current master wants to leave the bus, arbiter give access
75         to another master.
76         if(HBUSREQ(actual_master)='0') then
77             HGRANT(actual_master)='0';
78         end if;
79         if(HBUSREQ='0') then
80             for j in 0 to Ncores-1 loop
81                 if(HBUSREQ(j)='1') then
82                     HGRANT(j)='1';
83                     actual_master<=j;
84                     s1<=s1;
85                     exit;
86                 end if;
87             end loop;
88         end if;
89         else
90             s1<=s0;
91         end if;
92         when others=>
93             s1<=s0;
94     end case;
95     end if;
96     end if;
97     end process;
98
99     -- HMASTER generation --
100    -- This signal indicates the current master that has the bus.
101    process(HCLK)
102       begin
103           if(HCLK='1' and HCLK'event) then
104               if(RESET='0') then
105                   -- reset
106                   previous_master<=0;
107                   HMASTER<=(OTHERS=>'0'); -- HMASTER = MASTER0
108               else
109                   if(actual_master/=previous_master) then
110                       if(HREADY='1') then
111                           previous_master<=actual_master;
112                   end if;
113               end if;
114       end process;
On-chip bus standards in a broadcast architecture

arbiterAHB.vhd

```vhdl
113   HMASTER<=conv_std_logic_vector(actual_master, HADDRbits);
114   end if;
115   end if;
116   end if;
117   end if;
118   end process;
119   end Behavioral;
120
121
```

Page 3
On-chip bus standards in a broadcast architecture

---

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.MyDataTypes.all;
use IEEE.Numeric_STD.ALL;

entity decoderAHB is
  PORT(
    SEL : IN STD_LOGIC_VECTOR(HADDRbits-1 DOWNTO 0);
    OUTPUT : OUT STD_LOGIC_VECTOR(NCores-1 DOWNTO 0)
  );
end decoderAHB;

architecture Behavioral of decoderAHB is
begin
  process(SEL)
  begin
    for i in OUTPUT'RANGE loop -- SEL range is HADDRbits-1 downto 0
      if (conv_integer(SEL)=i) then -- Bin-to-Integer conversion
        OUTPUT(i)<='1';
      else
        OUTPUT(i)<='0';
      end if;
    end loop;
  end process;
  end Behavioral;
On-chip bus standards in a broadcast architecture

---

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package MyDataTypes is

-- ********** Constants ********** --
constant Ncores : integer := 8;
constant HADDRbits : integer := 3; -- LOG2(Ncores)
constant NumDataBits : integer := 8;
constant Ntransfers : integer := Ncores-1;
constant MaxRange : integer := 20;
constant STEP_DELAY : time := 21651 fs;

---

type data_vct is array (integer range <>) of std_logic_vector(NumDataBits-1 downto 0);
type haddr_vct is array (integer range <>) of std_logic_vector(HADDRbits-1 downto 0);
type twobit_vct is array (integer range <>) of std_logic_vector(1 downto 0);

-- DATA OF CORES -- (data for 8 modules)
constant MEM : data_vct(Ncores-1 downto 0) := (x"80", x"40", x"20", x"10", x"08",
x"06", x"02", x"01");

---

-- Declaration of function "region" --
function region (numCore: integer) return integer;
end MyDataTypes;

package body MyDataTypes is

-- Function region --
function region (numCore: integer) return integer is

variable reg: integer;

begin

for i in 1 to MaxRange loop

if(numCore<2*i*(1+i)) then

reg:=i;

exit;

end if;
end loop;

return reg;

end region;

end MyDataTypes;