**Title**

HW-SW SystemC Co-Simulation SoC Validation Platform

**Interconnect Methodology Summary**

Contract No. 4200022968

by

 TU Braunschweig, Germany

September 9, 2012

**Document No.: IDA-SCSV-IMS-001**

**Table of Contents**

1 Introduction 4

1.1 Purpose and Scope 4

1.2 Revisions 4

2 AMBA interconnect 5

2.1 AHB Modeling 5

2.1.1 AHB Payload 5

2.1.2 AHB Protocol mapping 5

2.1.3 Not or partially supported features 8

2.1.4 Creating/Binding AHB sockets 8

2.2 APB Modeling 10

2.2.1 APB Payload 10

2.2.2 APB Protocol mapping 10

2.2.3 Creating/Binding APB sockets 10

2.3 AXI Modeling 11

2.3.1 AXI Payload 11

2.3.2 AXI Protocol mapping 11

3 Processor sockets 11

4 SignalKit Sockets 11

**Table of Tables**

Table 1 - Revisions of this document 3

**Table of Figures**

Figure 1 - AHB write transfer (single) 5

Figure 2 - AHB read burst 6

Figure 3 - AHB write burst 7

Figure 4 - APB blocking transport delay 9

# Introduction

## Purpose and Scope

This document is the Interconnect Methodology Summary of the SystemC Co-Simulation SoC Validation Platform (SoCRocket). It describes the IP interconnect infrastructure of the library.

## Revisions

The following table will be updated during the course of the project.

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 0.1 | 09/09/12 | Initial version |
|  |  |  |
|  |  |  |
|  |  |  |

Table 1 - Revisions of this document

# AMBA interconnect

Almost all components of the library are equipped with an AMBA interface. At TLM level these interfaces are represented by sockets, which are customized for interchanging payload of a certain format. The AMBA sockets used for modeling the SoCRocket communication interfaces are supplied by a TLM AMBA Modeling Kit, which has been developed under the roof of GreenSoCs. The kit is distributed by Carbon Design Systems Inc. and can be freely downloaded and used. For instructions please see the SoCRocket User Manual.

## AHB Modeling

### AHB Payload

### AHB Protocol mapping

#### LT Abstraction

At LT abstraction AHB transfers are modeled using plain TLM 2.0 blocking transfer calls (b\_transport). The initiator starts the transaction at the beginning of the AHB address phase. The delay of the interconnect components and the target is aggregated in the transaction and returned to the initiator. The initiator is responsible for synchronization and may or may not decide to run ahead of time.

#### AT Abstraction

At AT abstraction AHB transfers are modeling using four timing points. The timing points relate to the phases to the TLM 2.0 standard protocol. However, in order to model timing in a more accurate way one additional phase transition was required. In following we will explain all relevant use cases in detail.

**AHB write/read transfer (single)**

Figure 1 shows the TLM phase assignment for a single-beat write transfer:

* Initiator sends BEGIN\_REQ at beginning of AHB address phase
* Target sends END\_REQ at the end of the AHB address phase (address sampled by slave)
* Target sends BEGIN\_RESP at the beginning of the AHB data phase; corresponds to HREADY becoming high (slave ready - wait states over). The delay of BEGIN\_RESP should also reflect the wait states produced by the target during the data phase (not just the once before accepting the first data item).
* Initiator sends END\_RESP at the end of the AHB data phase. END\_RESP must be delayed by the number of cycles VALID was low (transmission delayed by master).



Figure 1 - AHB write transfer (single)

**AHB read burst**

Figure 2 shows the phase protocol mapping for an AHB read burst:

* The initiator sends BEGIN\_REQ at the beginning of the AHB address phase
* The target will send BEGIN\_RESP at the beginning of the AHB data phase.  
  The number of wait states is delaying BEGIN\_RESP. This includes both: the wait states before delivering the first data item and the intermediate wait states inserted during transfer (target blocking).
* Because address phase and data phase overlap during an AHB burst transfer, END\_REQ will usually be send after BEGIN\_RESP. END\_REQ marks the end of the AHB address phase. It relates to the time when the target samples the last address of the burst.
* The initiator sends END\_RESP at the end of the AHB data phase. END\_RESP will be delayed by the number of initiator stall cycles (VALID low).



Figure 2 - AHB read burst

**AHB write burst**

The synchronization points of AHB write bursts are similar to AHB read bursts. The full mapping is shown in Figure 3:

* The initiator sends BEGIN\_REQ at the beginning of the AHB address phase.
* The target sends BEGIN\_RESP at the time the first data item is sampled, delayed by the total number of wait states involved in the transactions (cycles HREADY is low). This includes both: the wait states before delivering the first data item and the intermediate wait states inserted during transfer (target blocking).
* Because address phase and data phase overlap during an AHB burst transfer, END\_REQ will usually be send after BEGIN\_RESP. END\_REQ marks the end of the AHB address phase. It relates to the time when the target samples the last address of the burst.
* The initiator sends END\_RESP at the end of the AHB data phase. END\_RESP will be delayed by the number of initiator stall cycles (VALID low).

****

Figure 3 - AHB write burst

### Not supported or partially supported features

The model does not support early burst termination.

In case of a data split the slave is supposed to delay BEGIN\_RESP. The arbiter (ahbctrl) will not use the additional delay to schedule another master.

### Creating/Binding AHB sockets

The easiest way to create custom components with AHB sockets is to inherit from one of the AHB modeling base classes: AHBMaster, AHBSlave. The procedure is explained in detail in the SoCRocket User Manual.

For manual instantiation include header amba.h from Carbon TLM AMBA modeling kit.

**Master socket declaration (single):**

amba::amba\_master\_socket<32> ahb;

**Instantiation (module constructor):**

ahb("ahb", amba::amba\_AHB, ambaLayer\*, false\*\*);

\*ambaLayer = amba::amba\_AT or amba::amba\_LT

\*\* Only used for CT modeling

**Slave socket declaration (single):**

amba::amba\_slave\_socket<32> ahb;

**Instantiation (module constructor):**

ahb("ahb", amba::amba\_AHB, ambaLayer\*, false\*\*);

\*ambaLayer = amba::amba\_AT or amba::amba\_LT

\*\* Only used for CT modeling

**Multi-master socket declaration:**

amba::amba\_master\_socket<32, 0\*> ahbIN;

**\*** Number of channels for this socket - 0 meaning no limit

**Instantiation (module constructor):**

ahbIN("ahbIN", amba::amba\_AHB, ambaLayer\*, false\*\*)

\*ambaLayer = amba::amba\_AT or amba::amba\_LT

\*\* Only used for CT modeling

**Multi-slave socket declaration:**

amba::amba\_slave\_socket<32, 0\*> ahbOUT;

\* Number of channels for this socket - 0 meaning no limit

**Instantiation (module constructor):**

ahbOUT("ahbOUT", amba::amba\_AHB, ambaLayer\*, false\*\*)

\*ambaLayer = amba::amba\_AT or amba::amba\_LT

\*\* Only used for CT modeling

**Binding sockets:**

Binding AHB sockets follows the rules for TLM2.0 socket binding.

initiator.ahb(target.ahb)

See ./platforms/leon3mp/sc\_main.cpp for more examples.

## APB Modeling

### APB Payload

### APB Protocol mapping

The APB protocol is intended for low-bandwidth communication with I/O components or memory mapped control registers. In contrast to AHB or AXI, APB is not pipelined and can therefore be sufficiently modeled using blocking communication.

Figure 4 shows a simple APB write transfer and its abstraction using a blocking transport call:

* The initiator calls b\_transport at T2.
* The target will instantly return the aggregated component and setup delays (2 cycles).
* The initiator receives the delay a blocks the bus.
* The next transaction may not be issued before T4.



Figure 4 - APB blocking transport delay

### Creating/Binding APB sockets

The easiest way to create custom APB components is to instantiate a GreenReg socket. GreenReg sockets are extended AMBA sockets, which allow the specification of register interfaces.

Please include the amba.h header of the Carbon AMBA TLM Modeling Kit and the greenreg\_ambasockets.h header in your design.

A detailed description on how to set up APB components and register interfaces is given in the SoCRocket library user manual.

**Master socket declaration (single):**

gs::reg::greenreg\_socket<gs::amba::amba\_slave<32> > apb;

**Instantiation (module constructor):**

apb("apb",   
 r, // Name of GreenReg register container  
 APBDevice::get\_base\_addr\_(), // APB base address  
 APBDevice::get\_size\_(), // APB size of address space  
 amba::amba\_APB, // Bus protocol specifier  
 amba::amba\_LT, // APB is always LT (blocking)  
 false), // Only used for CT modeling

## AXI Modeling

### AXI Payload

### AXI Protocol mapping

# Processor sockets

# SignalKit Sockets