AN OPEN-SOURCE VHDL IP LIBRARY WITH PLUG&PLAY CONFIGURATION

Jiri Gaisler

Gaisler Research, Första Långgatan 19, 413 27 Göteborg, Sweden

Abstract: An open-source IP library based on the AMBA-2.0 AHB/APB bus is presented. The library adds “plug&play” capability to the AMBA bus, while maintaining compatibility with existing AMBA cores. The library is both vendor and technology independent, and has support for several commercial tool-chains.

Key words: IP cores; VHDL; system on chip; SOC; plug&play; SPARC; AMBA; PCI

1. INTRODUCTION

To shorten the development time of complex system-on-chip (SOC) devices, IP-core based design methodology is increasingly being used. However, integrating third-party cores from different suppliers can require significant adaptation and harmonization of both functional and logistical (tool-oriented) interfaces. Procurement and licensing conditions, NDA agreements, and royalty issues must also be resolved, and can be equally challenging.

One approach to solve these problems has been through the development of open-source IP cores, such as the LEON SPARC processor [1] (ESA & Gaisler Research) or the many cores developed by OpenCores.org [2]. While these open-source developments are interesting and promising on their own rights, they do not solve the combined functional and logistical problems with third-party IP reuse as described above. The procurement and licensing issues are simplified, but several technical issues persist. The LEON processor consists of a set of IP cores with harmonized interfaces and simulation/synthesis scripts, but the model implements only one major

function (processor). The OpenCores group provide many different cores, but neither the functional nor logistical interfaces are well harmonized.

The development of the GRLIB open-source IP library is a new attempt to address the reuse problems. The library consists of configurable IP cores with common interfaces and package structure. This paper will describe the rational and some implementation details of this new library.

2. GRLIB CONCEPTS

2.1 Overview

The GRLIB IP Library intends to enhance the development of SOC devices by providing reusable IP cores with common functional and logistical interfaces. Common functional interfaces means that the cores implement the same on-chip bus interfaces, and use the same configuration and technology mapping methods. Common logistical interfaces means that the IP cores are coherently packaged in a common distribution, and use the same scripting method for compilation and synthesis.

A key aspect has been to insure vendor independence at both core and tool level. The library is designed to be easily portable to different CAD tools and target technologies. It does also not depend on a vendor specific inter-face or technology which needs to be licensed or procured. The library is also designed to allow contributions or extensions from other parties. Each contributor is assigned a unique vendor name and ID, and can add cores to a well defined and separated part of the library.

The overall concept of GRLIB is thus to provide a standardized and vendor-independent infrastructure to deliver reusable IP cores. The sections below will outline how the various parts of this infrastructure has been implemented.

2.2 Library organization

GRLIB is organized around VHDL libraries, where each IP vendor is assigned a unique library name. Using separate libraries avoids name clashes between IP cores and hides unnecessary implementation details from the end user. A library typically contains a number of packages, declaring the exported IP cores and their interface types. All vendor-specific files such as simulation and synthesis scripts are contained in a vendor-specific area, and automatically merged into a combined script by a global makefile. Adding and removing of libraries and packages can thus be made without modifying

any global files. This also insures that modification of one vendor's library cannot will not affect other vendors.

The initial release of GRLIB can generate scripts for the Modelsim simulator, and the Synopsys, Synplify and Xilinx XST synthesis tools. Support for other CAD tools can easily be added.

2.3 On-chip bus

The GRLIB is designed to be “bus-centric”, i.e. it is assumed that most of the IP cores will be connected through an on-chip bus. The AMBA-2.0 [3] AHB/APB bus has been defined as the common on-chip bus, due to its market dominance (ARM processors) and because it is well documented and can be used for free without license restrictions.

GRLIB uses VHDL record types to group related bus signals. Each AMBA unit only uses two record signals to connect to the bus, one record with all inputs and one record with all outputs. The AMBA bus is multiplexed (Figure 1), and each AMBA unit drives a separate record signal to the arbiter, which selects the active unit to drive the shared bus.

Figure 1. AMBA AHB Bus Multiplexers

2.4 Distributed address decoding

Adding an IP core to the AHB bus is however not as straight-forward as just connecting the bus signals. The address decoding of AHB is centralized, and a shared address decoder and bus multiplexer must be modified each time a core is added or removed. To avoid dependencies on a global resource, and allow SOC design to be as simple as putting together LEGO bricks (!), distributed address decoding has been added to the GRLIB cores and AMBA AHB/APB controllers.

In a standard AMBA system, a fixed address decoder generates a chip select for each AMBA slave. In the GRLIB AMBA system, each instantiated core is assigned an address space through VHDL generics, and this information is routed to the address decoder together with the other bus signals in the output record. The address decoder does not have a fixed chip select generator, but automatically builds the chip select logic from the information received from the slaves. In this manner, the chip select generator always corresponds to the current configuration! Since the address space information is passed through VHDL generics, they will be constant and allow an efficient synthesis of the chip select logic. Using generics also allows an IP core to be instantiated several times with different parameters.

The example below shows two instantiation of the same UART device:

```
uart1 : apbuart -- UART 1
generic map (apbndx => 1, apbaddr => 16#200#, irq => 3)
port map (rstn, clk, apbi, apbo(1), uartin1, uartout1);

uart2 : apbuart -- UART 2
generic map (apbndx => 2, apbaddr => 16#300#, irq => 4)
port map (rstn, clk, apbi, apbo(2), uartin2, uartout2);
```

2.5 Interrupt steering

A problem with AMBA and most other on-chip buses is that interrupt steering is not defined in the bus specification, and must be solved in a system-dependent ad-hoc manner. GRLIB therefore provides a unified interrupt handling scheme by adding 32 interrupt signals to the output record of the AMBA modules. Similar to the address space definition, a VHDL generic is used to select which interrupt the unit should drive. The remaining interrupt signals are simply driven with zero. The arbiter ORs the 32-bit interrupt vector from each unit together into a combined 32-bit vector, and sends it out on the input records. The AMBA unit that implements an interrupt controller can then monitor the combined interrupt vector and generate the appropriate processor interrupt. In this way, interrupts can be generated regardless of which processor or interrupt controller is being used in the system, and does not need to be explicitly routed to a global resource. The scheme also allows that interrupts are shared by several cores and resolved by software.

2.6 Plug&Play capability

A broad interpretation of the term “plug&play” is the capability to detect the system hardware configuration through software. Such capability makes it possible to use software application or operating systems which

automatically configure themselves to match the underlying hardware. This greatly simplifies the development of software applications, since they do not need to be customized for each particular hardware configuration.

The information which is needed for plug&play capability includes the type of cores that are attached to the local bus, how they are configured, which address range they are mapped to, and which interrupt they drive. In the well-know PCI [4] bus, this information is stored in the “configuration space” and accessed through special configuration read/write cycles.

In order to bring “plug&play” capability to AMBA, functionality to access the configuration parameters is needed. This functionality must be provided without modification to the basic bus protocol, or compatibility with the large infrastructure of AMBA compliant third-party IP cores would be lost. This rules out the addition of new bus cycle types, such as the configuration read/write cycles on PCI.

In GRLIB, the address mapping configuration is sent from all cores to the AMBA address decoder, where it is used for chip select generation. By providing the remaining plug&play information (core type and interrupt steering) in the same way, the address decoder contains the full system configuration. The decoder is designed to automatically concatenate the configuration information from all AHB devices are into a small configuration prom, by default placed at the top of the address space. Any AHB master can then read the system configuration using standard read cycles, and a plug&play operating system can be supported.

To provide the plug&play information from the AHB units in a harmonized way, a configuration record for AMBA AHB devices has been defined (Figure 2). The configuration record consists of 8 32-bit words, where four contains configuration words defining the core type and interrupt routing, and four contain so called “bus address registers” (BAR), defining the memory mapping.

The configuration word for each device includes a vendor ID, device ID, version number, and interrupt routing information. The BARs contain the start address for a segment allocated to the device, a mask defining the size of the segment, and information whether the segment is cacheable or pre-fetchable. The configuration record can contain up to four BARs and the core can thus be mapped on up to four distinct memory segments.

By identifying the cores through a vendor and device ID, a mapping between the VHDL library assignment and vendor ID can be established. Each vendor receives a unique vendor ID, and can add cores and device IDs without conflicts with other vendors.

Figure 2. AMBA AHB configuration record

2.7 Portability

Mapping IP cores to the desired target technology is normally handled by the synthesis tool, but does not work for technology specific mega cells such as embedded memories, hard multipliers or pads. GRLIB is intended to be technology independent, and therefore provides “portability” API for these type of blocks. The API is implemented as a component with a VHDL generic to select the target technology. In the architecture of the component, VHDL generate statement are used instantiate the corresponding macro cell from the selected technology library. For RAM cells, generics are also used to specify the address and data width, and the number of ports. Below is an example of a pad and a single-port ram with technology selection.

```
entity outpad is
  generic (tech, level, slew, voltage, strength : integer := 0);
  port (pad : out std_ulogic; i : in std_ulogic);
end;

entity syncram is
  generic (tech : integer := 0; abits, dbits : integer := 8 );
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    enable   : in std_ulogic;
    write    : in std_ulogic);
end;
```

The portability API includes components such as single-port ram, two-port-ram, dual-port ram, single-port rom, clock generators and pads.

2.8 Availability

A working beta version of GRLIB has been available since early 2004. The library includes cores for AMBA AHB/APB control, the LEON3 SPARC processor, 32-bit PC133 SDRAM controller, 32-bit PCI bridge with DMA, 10/100 Mbit ethernet MAC, 8/16/32-bit prom and sram controller, generic UART, modular timer unit, interrupt controller, and a 32-bit GPIO port. Memory and pad generators are available for Virage, Xilinx, UMC, Atmel, and Actel.

The first public version will be made available during Q2 2004, and released under GNU GPL.

3. CONCLUSIONS

GRLIB has been developed to simplify both the development and reuse of open-source IP cores. By specifying common functional interfaces, new cores can easily be added and immediately reused. The logistical interface support several commonly used CAD tools, and can easily be extended. The distributed address decoding allows to design and modify complex SOC designs without having to modify any global controller or decoder. To enhance portability, an API to map technology-specific macro cells has been defined. To simplify software development and debugging, a plug&play capability has been added. By organizing GRLIB according to vendors, modification or addition of IP cores can be done without side-effects. By providing the library freely under the GNU GPL license, immediate access is provided to both academia and commercial companies.

REFERENCES

- [1] LEON2 home page, <http://www.gaisler.com/leon.html>
- [2] Opencores home page, <http://www.opencores.org>
- [3] AMBA Specification 2.0, ARM Ltd, May 13, 1999, <http://www.arm.com>
- [4] PCI Local Bus Specification, Revision 2.1, June 1, 1995, PCISIG