

A High-level Design and Implementation Platform for IP Prototyping on FPGA

T.G.R. van Leuken, A.C. de Graaf and H.J. Lincklaen Arriëns
CAS Section, Department of Micro-Electronics,
Faculty of EEMCS, Delft University of Technology,
Mekelweg 4, 2628 CD Delft, the Netherlands
E-mail: T.G.R.vanLeuken@EWI.TUdelft.NL

Abstract— **The purpose of this platform is to provide a design and implementation environment for prototyping both computational and controller functions, in a standard communication architecture, including an Atmel AVR micro-controller and a Wishbone bus protocol. This platform serves the goal of the fast implementation and test of a data-path or controller IP blocks in an FPGA. The purpose of the proposed design flow is to support a behavior design and implementation flow. The design flow consists of a scheduling and mapping tool and a number of free and commercial EDA tools.**

Keywords— **Communication; IP; behavioral design; data-flow; FPGA**

I. INTRODUCTION

To assist a designer –in our case particularly MSc students as part of their laboratory work– in his/her attempts to convert a behavior level description [2] of a ‘function’ to be implemented in hardware (IP block), we have developed an EDA program, which is capable of scheduling and mapping simple operations on hardware resources. These operations are currently limited to ALU functions like multiplication, subtraction, comparison and addition. However, since many computational and signal processing functions consist of only these functions, many of them can be implemented.

To provide for a standard communication interface/architecture and programming environment, instead of using the complex and heavily loaded 32 bit platform described before [1], we have augmented the freely available VHDL soft-core [3] of an Atmel AVR micro-controller [6], with a Wishbone bridge. Many Wishbone [4] compliant IP cores are already available and can easily be integrated into this platform. The Atmel AVR micro-controller gives the designer the additional benefit of user-friendly software programming environments [5] for both Windows and Linux.

To provide for a ‘classroom’ laboratory design environment, we have additionally designed a Wishbone slave IP interface, allowing for easy integration of VHDL blocks to be designed by the MSc students. After scheduling and mapping the operations of a ‘student function’ in time and on resources, the EDA program generates the standardized interface as VHDL entities and component statements. The usage of this interface has the additional advantage of making available to the designer VHDL test-bench template files to be used in a simulation environment. Finally, the design can be tested on a Xilinx FPGA development board.

II. HARDWARE ARCHITECTURE

A. Atmel AVR

We have opted to use the Opencores AVR core for several reasons. First, we want to have full control over our design, hence we need VHDL code. Secondly, since we want to concentrate on the design of ‘simple’ IP blocks and IP communication, we do not need a complex CPU block. Finally, we want to have a supported and good quality C compiler available. The Opencores AVR core provides for all these needs.

The VHDL model of the AVR [6] is an 8-bit micro-controller based on the ATMEL AVR RISC architecture. The AVR executes its instructions in a single clock cycle, achieving throughputs approaching 1 MIPS per MHz. The AVR core is based on an enhanced RISC architecture that combines an instruction set with 32 general-purpose working registers. All 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. Memory access is based on a Harvard architecture (Figure 1).

The implementation of the VHDL model contains the basic elements of the ATmega103 on which it is based.

Except for the 32 general-purpose working registers, it contains a 16-bit timer, 4 level and 4 edge triggered interrupts, and a UART. The general-purpose I/O lines (ports) are configurable. The Atmega103 instruction set is fully supported.

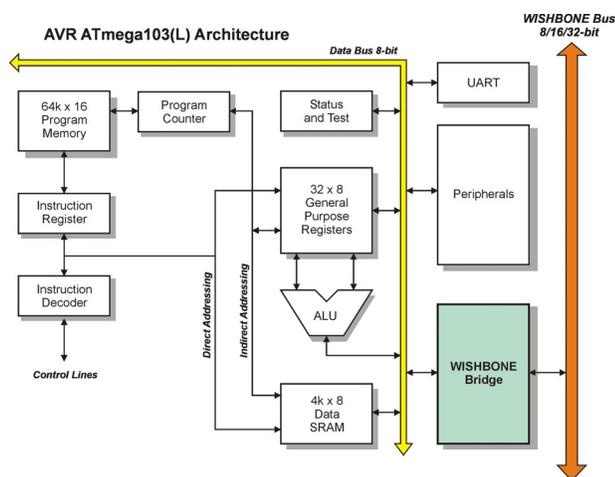


Figure 1. Architecture of the AVR core including the Wishbone Bridge.

B. The Wishbone Bus

The Wishbone bus is a simple scalable bus specification to connect IP blocks. The specification of the bus is in the public domain. As a consequence many IP blocks have been developed using this type of bus interface and many are available. All Wishbone bus data transfers can execute in one clock cycle. It can be configured as an 8, 16 or 32 bit wide bus. All bus cycles use a handshaking protocol between the master and the slave IP block. The architecture of the bus is not defined; it is up to the user/designer to choose one. We have selected a shared bus architecture.

C. Wishbone Bridge

The Wishbone bridge is responsible for the transfer of data and address values between the AVR bus and the Wishbone bus. It also generates the Wishbone control signals. The bridge contains some memory registers to store the address, data in, data out and control values (Figure 2).

There are two methods of connecting the bridge with the AVR. The first option would be to connect the general-purpose I/O lines of the AVR with the registers of the bridge. The advantage of this method is that the bridge is fully software controllable. As a consequence, this makes it also slow.

The second method is to connect the bridge with the internal AVR bus. In our case, this option exists because not all AVR registers are used. Since the AVR implementation does not contain the AD converter, the AD control registers are in fact free. By simply mapping these registers to the Wishbone bridge registers we have created a simple and fast interface between the AVR and the bridge. The bridge also generates Wishbone control signals using a state machine, which is triggered by the AVR control lines, the read and write signal. The problem to map the timing and semantics of the AVR control signals to the Wishbone signals has been solved here. Figures 3 depict the basic read and write operation of the Wishbone bus and C code functions.

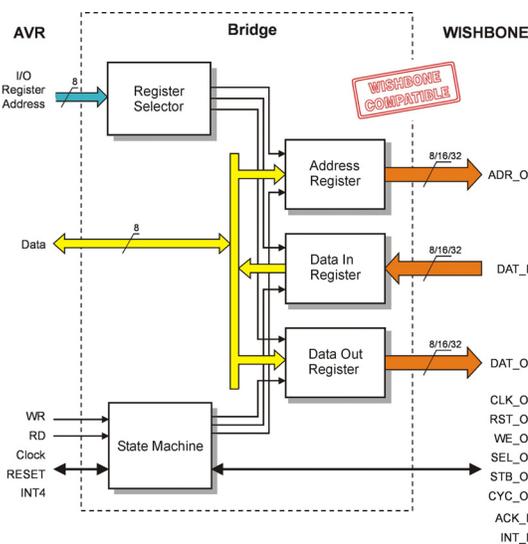


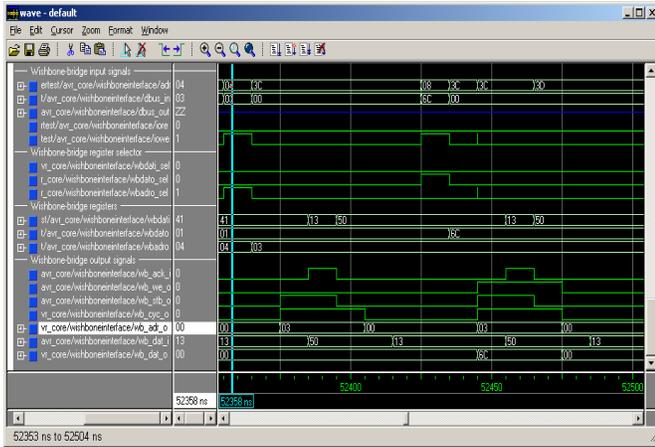
Figure 2. The Wishbone bridge architecture.

III. 'IP' SLAVE BLOCKS

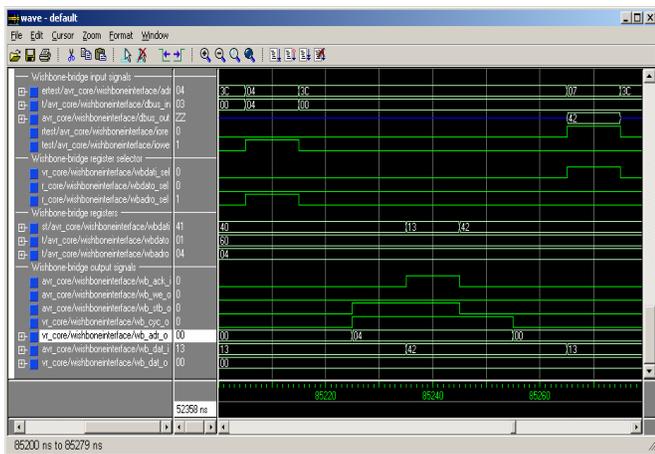
A. I/O slave blocks

To provide for basic communication with the outside world, we have developed or adapted several IP slave blocks, including an I2C, an USB, and a Codec interface. Several other interfaces are under development by students as a part of their MSc course laboratory activities. These include e.g. a VGA controller, an IDCT and an SPI interface.

The Codec interface is important to us, since one of the exercises for the students is to develop an audio filter.



a) Wishbone bridge write cycle



b) Wishbone bridge read cycle

```
void wb_write(unsigned char adr, unsigned char data)
{
    WBADR = adr;
    ...

    WBDATO = data;
}

unsigned char wb_read(unsigned char adr)
{
    unsigned char dati;

    WBADR = adr;
    ...

    dati = WBDATI;
    return dati;
}
```

c) C-code for the read and write functions.

Figure 3. This figure shows the basic read and write operations of the Wishbone bus, a) Simulation window of a read operation, b) Simulation windows of a write operation, c) C-code of the basic read and write function.

Here it is important to note that all IP blocks reflect the available resources on a specific FPGA development board, i.e. our codec implementation is tailored for a particular type of codec chip.

B. The student IP block

For the 'classroom' laboratory design environment, we additionally defined a Wishbone slave IP interface, allowing for easy integration of the VHDL blocks to be designed.

We provide a layer (wb student interface) that connects the AVR to the student IP block. In this layer data from/to the AVR is stored in local memory and two control lines are available for, first, signal the student IP block to start, and second, to signal the AVR when the student IP block is done. The two memory blocks of 127 bytes each are used to store and retrieve the input and output variables of the student IP block. In principle any word size can be selected. For example, 31 4-byte integers could be used. The selection of the word size has as a consequence that the C program that runs on the AVR, needs to be adapted in such a way that it addresses the correct memory content.

After scheduling and mapping the operations part of the student block in time and on resources, the EDA program (see section IV) generates the standardized interface of the student IP block as VHDL entities. The requirement to use this interface has the additional advantage of making available to the designer VHDL test-bench template files, which can be used for simulation.

IV. TOOLING

A. Methods/Approaches: Scheduling and Mapping

To assist the designer (student) in his/her attempts to convert a behavior level description of a 'function' (IP block) to be implemented in hardware, we have developed an EDA program, which is capable of scheduling and mapping simple operations on hardware resources.

These operations are currently limited to some ALU functions, such as multiplication, addition, subtraction and comparison. The tool lets designers enter a circuit description (Figure 5) and after specifying some attributes like type and number of resources, the tool creates a scheduled Data Flow Graph (DFG) (Figure 6) and maps the operations to resources.

The tool is set up as a collection of (Matlab) functions, some of which are accessible through a GUI. The functions can be used for testing an algorithm both in the Matlab environment as a reference, as well as for

supplying the VHDL test benches. For displaying the flow graphs, we have chosen the ‘dot’ layout tool from AT&T’s GraphViz project [7].

Currently, students can choose from the ASAP and ALAP scheduling methods (minimum number of clock states, unlimited resources), a Force Directed scheduling method [8] (minimum number of clock states, minimum number of resources which are optimally distributed over the available clock states) and a List scheduling method (user defined number of resources that determine the number of clock states needed).

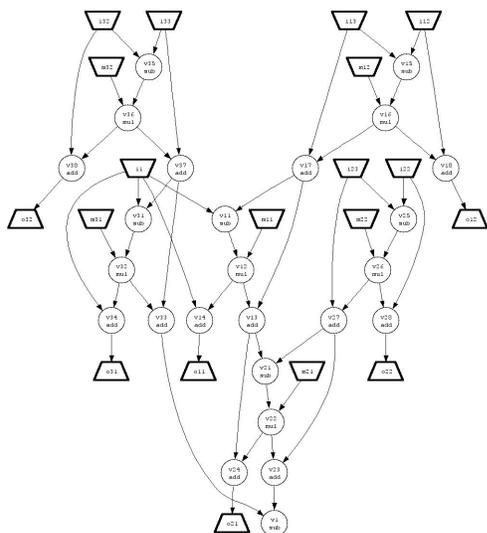


Figure 4. A DFG of an example (6th order Lattice Wave Digital Filter) students may use to experiment with scheduling tools and implementation.

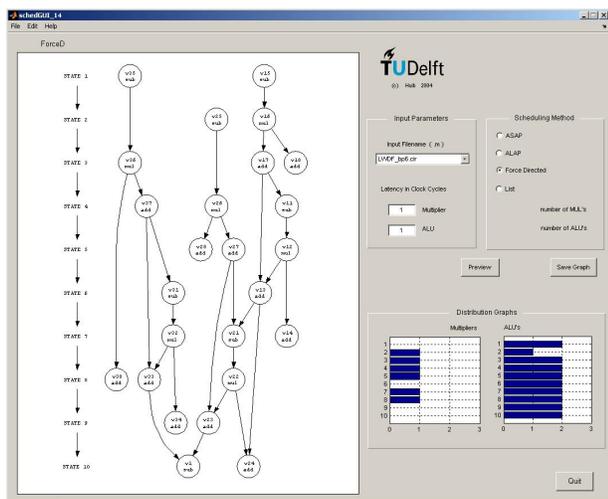


Figure 5. The user interface of the scheduling and mapping tool.

B. Free and commercial EDA tools

We use a number of free tools for implementing the VHDL design on the FPGA. Our Xilinx boards facilitate the use of the Xilinx WebPack Package. To support the software design flow, the WinAVR or GCCAVR C development tool set is employed.

In some situations these free hardware design tools may not be capable of handling the size of the design. In such a case we use commercial tools to support part of our design flow.

V. RESULTS

The resulting environment consists of a design flow, which takes the designer from abstract function descriptions, to a scheduled and mapped description.

Given the generated standard Wishbone interface, the implementation of the function in a complete SoC environment has become straightforward.

The platform environment allows for fast and simple integration of Wishbone compatible, scalable IP blocks. The scalable Wishbone bridge (8/16/32/bit) is part of the AVR register IO map, hence software programmable. Currently, a shared bus implementation, and several slave IP blocks, such as Codec, SPI and I2C modules are available.

The number of logical gates needed is approx. 150,000 gates and the maximum realizable CPU clock speed is typically 20MHz on a Xilinx Virtex-family FPGA.

REFERENCES

- [1] Proceedings FPL 2002, Montpellier, France, September 2002. "General Purpose Prototyping Platform for Data-Processor Research and Development", Filip Miletic, Rene van Leuken and Alexander de Graaf, 1187 pag.
- [2] "Synthesis and Optimization of Digital Circuits", De Micheli, ISBN 0-07-016333-2.
- [3] <http://www.opencores.org>, AVR VHDL core.
- [4] <http://www.silicore.net/wishbone.htm>, Wishbone Bus Specification.
- [5] <http://www.avrfreaks.net/AVRGCC>, WinAvr C compiler.
- [6] <http://www.atmel.com/avr>, Atmel AVR 8 bit RISC.
- [7] <http://www.graphviz.org>, GraphViz homepage.
- [8] P.G.Paulin and J.P.Knight, Force-Directed Scheduling for the Behavioral Synthesis of ASIC's, *IEEE Trans on CAD*, June 89, Vol 8, Nr 6, pages 661-679.