Automatic HDL Generation for ASIC Designs

JOUNI RIIHIMÄKI
Nokia
PO Box 88, FI-33721, Tampere, FINLAND

Abstract: - Documentation of a complex design is essential for the reuse and for the verification. Spreadsheet applications, such as MS-Excel, are often used to design and document certain parts of system-on-chip designs. For example interrupt and DMA connections are easy to describe in a table format. To facilitate the implementation, the actual HDL code can be automatically generated from the spreadsheet documentation. The benefit of this approach is that the HDL implementation is always in tact with the documentation and vice versa. The quality of design and documentation is also usually better when the manual coding is not needed. In addition, h-files and compilation scripts needed later in the flow can be generated along with the HDL code, which facilitates the use of the design.

This paper presents an approach and a tool to generate VHDL code and h-files from MS-Excel spreadsheet. The approach is presented with two examples: interrupt and DMA connections. The results show that the method dramatically reduces the time needed to the subsystem implementation.

Key-Words: - computer-aided design, code generation, quality, system-design.

1 Introduction

System design at high abstraction levels is often done utilizing tools that are not actually aimed for HW design, such as MS-Office products PowerPoint or Excel. They are used even if dedicated approaches for high-level design are available [1][2][3]. However, those tools are not yet widely used in commercial ASIC design since the quality of the generated code is not mature enough and/or the tools and their outcome cannot be seamlessly used with the utilized back-end flow.

Issues such as interrupt connections, DMA request connections, test connections, or I/O pin multiplexing are easy to define and document in a spreadsheet form. The details relating to such connections often change during the design cycle, especially if the design consists of several processors and DMA controllers, and if the I/O pin multiplexing is used to select a great number of functional signals. The implementation of those issues is, after all, straightforward task, but manual work and many iterations increase the risk of human error either to the design or to the documentation.

When the documentation gives an unambiguous description of the signal connections and multiplexer controls, the implementation of this part of the design can be automated. An automatic code generation from spreadsheet makes the design flow more smoothly, and it reduces number of bugs in the design dramatically. In addition, the documentation and implementation are automatically in tact with each other.

The code generation also improves the productivity since the designer can concentrate to the design work instead of error-prone manual coding. Once the code generation tool is available, the time required to generate the code can be neglected and the saved time can be used, for example, to the system work or to the verification.

The short implementation time also encourage innovation by enabling easy exploration to find out optimal solution. In addition, the automating code generation eliminated the temptation to use existing code in compromised solutions as discussed in [4]. However, often the generated code is not as optimal as hand made and, therefore, it is not suitable for timing or area critical parts of the design

In addition, the required h-files and compilation and synthesis scripts can be generated at the same time, which makes the SW developing less cumbersome since also the h-files are up-to-date. The compilation and synthesis scripts, in turn, improve the usability of the design.

In future, when the designs will contain even more functionalities, and thus more interrupts, DMA requests, and I/O signals, the approach will save even more time during the design process.

In this paper, we present a method to describe a system in MS-Excel and a tool to generate HDL description from that description. This paper is organized as follows: Chapter 2 describes the related works and Chapter 3 presents the implementation of the generation tool. The use of the tool is presented with two test cases as discussed in Chapters 4. After

that, the results are shortly discussed and the paper in concluded in Chapter 5.

2 Related Research

Automatic code generation offers several benefits as discussed in [4]. Even if the author handles the code generation from the SW point of view, the same aspects exists also in hardware implementation. The benefits of the automatic code generation are: the quality of the code is improved, the design time is reduced, and less verification effort is needed. For example code generated by Mathworks product is used in Nasa's X-43A Scramjet [5].

Approaches that use code generation in architectural exploration to try-out several different HW architectures are presented in [1] and [2]. In addition to [4], there exist many papers, which present code generation for different purposes and for different environments [6] and [7].

The code generator is part of many high-abstraction level design tools. For example, Telelogic Tau G2 UML design environment [4], which is used in [2], can generate application C code from the UML description. However, the HDL generation for HW implementation is often not supported. In [2], the HW part is generated with another self-made extension.

3 Tool Implementation and Usage

The basic principle in the code generation is simple. A generation tool gets the required information as an input; it converts the input data to some other format; and finally it gives the generated code out. For example, a standard C language compiler gets the input from the source code file(s), it translates the code to the binary format of the target platform, and the executable binary code is stored to a file.

The HDL code generation from spreadsheet documentation, for example from an MS-Excel description, is rather straightforward if the form of the document is beforehand agreed and it contains all the information required to implement the design.

The tool can be implemented with a scripting language such a Perl or TCL. We selected Perl for the implementation because it has been used in earlier projects and it has a very useful hash data type, which is missing from many other languages. The benefit of those scripting languages is that they are more efficient in ASCII file parsing that, for example C/C++, and typically the design time is shorten when a script language is used. However, the run time is often poorer when compared to a compiled program. Figure 1 depicts the basic

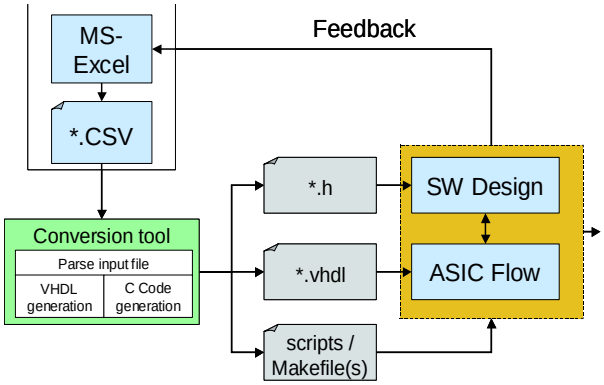

Figure 1. Usage of the tool.

structure of the tool and also the flow how the tool is used.

There exist several Perl libraries to read and modify MS-Excel binary files directly instead of converting the document to ASCII format. Spreadsheet::ParseExcel and Spreadsheet::WriteExcel libraries in Comprehensive Perl Archieve Network (CPAN) [10], for example, are freely available for that use. However, we wanted that the tool is usable in all workstation environments and with all MS-Office versions, and therefore we decided not to use those libraries.

Instead of MS-Excel's native binary format, we are using a comma separated ASCII file format (CSV) as an intermediate file. In CSV, the columns of the spreadsheet are separated with commas, and the lines are in different rows. The Excel worksheet can be saved to CSV and therefore it increases the portability of the tool and allows data sharing among other tools. The MS-Excel can be used to also open the CVS file. In addition, other spreadsheet applications can be used view and edit it as well as any text editor.

First in Figure 1, the design, e.g. interrupt connections, is described in an MS-Excel worksheet, which is saved in a CSV format. After that, a tool implemented with Perl is used to parse the file. The structure of the document is beforehand agreed and therefore the information can be easily found from the file. The tool stores all the essential information to internal data structure, which make use of Perls hash data structure to facilitate the data handling. When all data is stored and all the required dependencies are found, the HDL code is created.

In addition to the HDL code, the required h-files used by the SW can be generated as well as the compilation scripts. That ensures that those support properties are also always in tact with the design, and therefore increase the reusability.

The tool is modular and therefore it is easy to extend to generate, for example SystemC or Verilog code in addition to the synthesizable VHDL code. It

is also easy extend to support additional information that might be wanted to include to the spreadsheet.

The tool can also be linked together with a utilized version control system. In that case, the previous version of the module is checked out from the version control system before new versions are stored, new files are generated, and checked in. Also the Excel file is stored to the version control system. The creation date and the name of source file are included to the comments of the files. In addition it is indicated which version of the compilation scripts should be used. This information can be later used to identify the version of the module and find which version of the document is used when the code is generated.

4 Use Case: Interrupt and DMA Connections

In a multiprocessor system-on-chip design, there can be several interrupt controllers (one or more per processor) to handle the interrupts from the peripheral devices. Similarly, there can also several DMA controllers to handle all the DMA transfers. The division can be based on the thought use-cases, power usage scheme, or to the processor aimed to be used with the certain DMA request. Figure 2 shows the principles of the interrupt and DMA request connections.

Typically all the interrupts are not needed to be connected to all the interrupt controllers. This reduces the connections but, on the other hand, makes the design more challenging since the decision which interrupts are connected to which CPU affects to the SW design, and also to the possible use-case scenarios. Unavailable interrupt makes it impossible to execute certain application on the certain CPU. Naturally, the same stands also for the DMA request connections. Therefore it is probable that the connections will be iterated several times during the design cycle.

Moreover, all the IPs do not necessary follow the same scheme in interrupts. The polarity (active low or high) of a signal can be different and the interrupt type can be either pulse or level. In addition, in a complex design where several different clocks are used, the interrupts from different IPs may be synchronized to different clocks. If the pulse is too short compared to the target clock frequency the interrupt may not be noticed at all. Therefore it is required that the signal is synchronized with a suitable clock and converted to be same type to make sure that the interrupt is noticed by an interrupt controller.

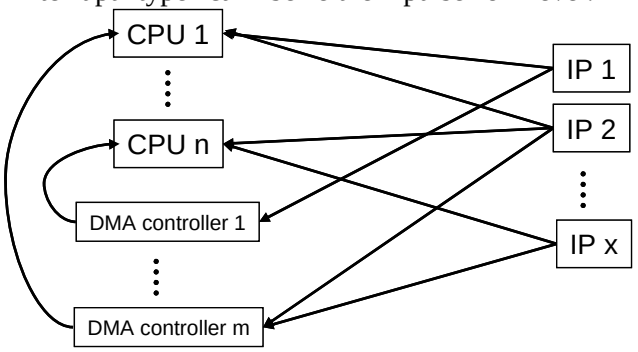

Figure 2. Principles of interrupt and DMA request connections

When the interrupts and DMA request are documented in a table format as shown in Table 1, the connections can be automatically created and implemented to the RTL code.

The spreadsheet document in Table 1 contains all the required information needed to generate the interrupt and DMA connections. First of all, the name and width of the signal is defined. After that, the source module is described in From column. The Connected to field describes where the signal is targeted. If all the bits are not connected, the connected bits are defined in parenthesis. The same signal can be connected to several targets. The Src Clock field defines the clock domain where the source module belongs (and if a synchronizer need to be added to he signal), and the last column defines if the signal needs to be inverted.

The code generation tool adds the inverter and synchronizer near the interrupt/DMA controller in the RTL if such a component is required. However, the physical locations of such modules naturally depend of the used back-end tools and given constraints.

Figure 3 shows an example of the created connections. To clarify the picture, all the other logic, clocks, bus connections are not shown. As can be shown of the Figure 3, the IP A belongs to different clock domain than the interrupt controller, the interrupt is synchronized to target clock.. In addition, the upper interrupt from IP A and interrupt from interrupt B is inverted. This is required, for example, if the interrupt controller recognizes rising edges and we are interested about falling edges.

Table 1. An example of interrupt connections description

Name	Width (bits)	From	Connected to	Src Clock	Inversion
IP_A.Interrupt 1	2	IP A	CPU 1(1:0); CPU n(0)	A	no
IP_A.Interrupt 2	1	IP A	CPU n	A	no
IP_B.Interrupt	1	IP B	CPU n	B	yes
IP_B.DMA req	3	IP B	DMA; CPU 1(0)	B	no
IP_C.Interrupt 1	1	IP C	CPU 1, CPU n	A	yes
…	…	…	…	…	…
IP_X.StatusSignal	4	IP D	DMA; CPU 1(1:0); CPU 2(3:2)	A	no

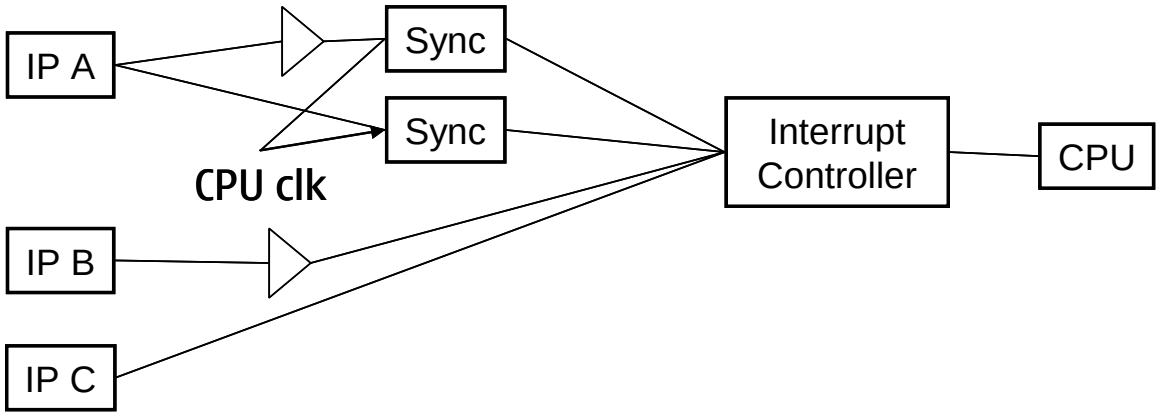

Figure 3. Interrupt connections, inversions, and synchronization

5 Conclusions

Two different use cases in a typical ASIC were presented. As discussed, the tool significantly reduces design time generating synthesizable VHDL code and required h-files and compilation scripts for the design sub-system presented in a spreadsheet document. The presented method also reduces the amount of required manual work and, especially in case when the implementation often changes, it also improved the quality of the design.

The area and timing of the generated modules are practically the same as if the modules would be implemented manually. However, this is expectable since the generated modules are after all very simple. The gain of this approach is that new implementation from updated spreadsheet document can be generated in a second. Moreover, for instance a changed interrupt connections does not cause any changes to the SW. This all saves at approximately five to ten hour implementation and verification work compared to a situation that all would be done manually.

In addition, time is saved with automatic code generation when manual coding is not needed. Therefore the saved time can be used to verification or to system-level design. This also improves the design quality since the designers can concentrate to details that are more essential.

The use of the method is presented with two examples. However, there are plenty of potential areas in system-on-chips where the approach can be used. In addition, the tool can be improved to generate, for example, SystemC code and required makefiles or compilation scripts for every design tool.

6 References

[1] Pimentel, P. Lieverse, P. van der Wolf, L. Hertzberger, and E. Deprettere, “Exploring embedded-systems architectures with Artemis”. IEEE Computer 34, 11, 2001, pp. 57-63.

[2] T. Kangas et al., “A Communication-Centric Design Flow for HIBI-based SoCs”, LNCS 3133 Computer Systems: Architectures, Modeling, and Simulation, A.D. Pimentel, S. Vassiliadis, (eds.), Springer-Verlag, Berlin, 2004, pp. 474-483.

[3] CoCentric System Studio, homepage of Synopsys, http://www.synopsys.com

[4] D. Maclay, “Click and Code (automatic code generation),”IEEE Review, Vol 46, Iss 3, May 2000, pp. 25-28.

[5] Homepage of mathworks Inc, http://www.mathworks.com

[6] M. Erba, R. Rossi, V. Liberali, A. G.B. Tettamanzi, “An Evolutionary Approach to Automatic Generation of VHDL Code for Low-Power Digital Filters”, Proceedings of the Genetic Programming European Conference, EuroGP 2001, LNCS 2038, Springer-Verlag, Berlin 2001, pp. 36-50.

[7] A. Valderrama, F. Nacabal, P. Paulin, and A. A. Jerraya, “Automatic Generation of Interfaces for Distrubuted C-VHDL Cosimulation of Embedded Systems an Industrial Experience”, Proceedings of 7th IEEE International Workshop on Rapid System Prototyping, June, 2001, pp. 72-77.

[8] Telelogic Tau G2, Homepage of Telelogic, http://www.telelogic.com

[9] J-M. Daveau, ”VHDL generation from SDL specications,” Proceedings of International. Conference. Computer Hardware Description Languages and Their Applications, Apr. 1997.

[10] Homepage of Comprehensive Perl Archive Network, www.cpan.org