Efficient Implementation of FPGA Based on Vivado High Level Synthesis

Huan Li
No.36 Research Institute of CETC
Jiaxing , Zhejiang 314033,China
e-mail: lh07054@126.com

Wenhua Ye[1,2]
[1] Science Technology on Communication Information Security Control Laboratory
[2] No.36 Research Institute of CETC
Jiaxing, Zhejiang 314033, China
e-mail: ye07054074@126.com

***Abstract*—The Xilinx Vivado High Level Synthesis (HLS) transforms a C specification into a register transfer level (RTL) implementation that can be synthesized into a Xilinx field programmable gate array (FPGA). It refers to the automatic integrated initially with C, C ++ or System C language to describe digital designs. Using HLS to explore all possibilities, analysis and performance characteristics of the area, and finalize a program to implement algorithms on FPGA chip. An example of a filter in this paper described the use of HLS to implement on FPGA quickly and efficiently.**

Keywords- FPGA; vivado; high level synthesis; C/C++ language

I. INTRODUCTION

The programming model of a hardware platform is one of the driving factors behind its adoption. Software algorithms are typically captured in C/C++ or some other high-level language, which abstracts the details of the computing platform. For the past few decades, the fast execution of algorithms captured in these languages have fueled the development of processors and software compilers. Although there are many kinds of specialized processors, such as the digital signal processor (DSP) and graphics processing unit (GPU), all of these processors are capable of executing an algorithm written in a high-level language, such as C, and have function-specific accelerators to improve the execution of their target software applications[1].

The software engineer must now structure algorithms in a way that leads to efficient parallelization for performance. The techniques required in algorithm design use the same base elements of FPGA design. The main difference between an FPGA and a processor is the programming model.

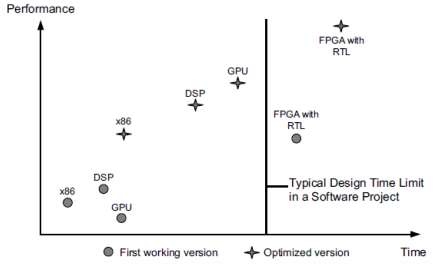

Figure 1. Design time vs. application performance with RTL design entry

Fig. 1 shows a traditional FPGA design flow with RTL as the design capture method, which illustrates how the programming model difference affects implementation time and achievable performance for different computation platforms [1].

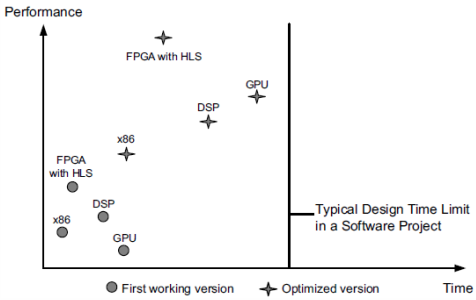

Figure 2. Design time vs. application performance with vivado hls compiler

Recent technological advances by Xilinx remove the difference in programming models between a processor and an FPGA. Just as there are compilers from C and other high-level languages to different processor architectures, the Xilinx Vivado High-Level Synthesis (HLS) compiler provides the same functionality for C/C++ programs targeted to Xilinx FPGAs. Fig. 2 compares the result of the HLS compiler against other processor solutions available to a software engineer [1].

The Xilinx Vivado High-Level Synthesis (HLS) tool transforms a C specification into a register transfer level (RTL) implementation that can synthesize into a Xilinx field programmable gate array (FPGA). Writing C specifications in C, C++, or SystemC, and the FPGA provides a massively parallel architecture with benefits in performance, cost, and power over traditional processors [2].

Engineers are so interested in high-level synthesis, not only because it allows engineers to work on a higher level of abstraction, but also it can easily generate a variety of design solutions. By using HLS, engineers can explore all possibilities, analysis and performance characteristics of the area, to finalize a program to implement algorithms on FPGA chip. For example, to explore the memory mapped to Block RAM (BRAM) or what have different effects on the distributed RAM, or analysis of loop expansion, and other related optimization circuits have any effect, but do not have to manually generate different register-transfer level (RTL)

978-1-4673-9026-2/16/$31.00 ©2016 IEEE

designs . What need to do is setting the relevant instructions C / C ++ / System C design only [3], [4].

Xilinx introduced the HLS tool in its latest release of Vivado ™ kit [1]. Vivado HLS is brand transformation and remodeling of AutoESL tool . It offers numerous techniques to optimize C / C ++ / SystemC code to achieve the target performance. This HLS tool can help engineers to quickly implement algorithms on the FPGA, without using RTL design method with very time-consuming hardware description language based on Verilog and VHDL . It works shown in Fig. 3[5], [6].

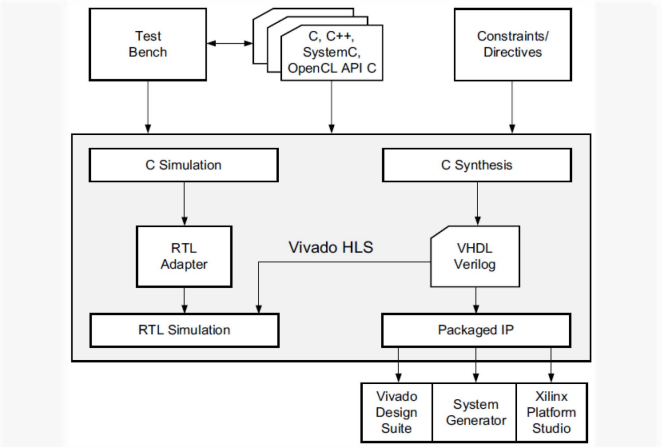

Figure 3. Vivado HLS design flow

II. IMPLEMENTATION METHOD

The Xilinx Vivado HLS tool synthesizes a C function into an IP block that can integrate into a hardware system. It is tightly integrated with the rest of the Xilinx design tools and provides comprehensive language support and features for creating the optimal implementation for your C algorithm [7], [8].

In a typical Vivado HLS process, it normally takes three C / C ++ files: source file (including C functions to be integrated), header file and testbench file described by calling main ( ) function [5].

Header include not only the source file that declares the function used, also includes instructions that support user-defined data type which has a specific bit-wide. Thus it allows designers to use different bit width that defined by the C / C ++ criteria wide. For example, the integer data type (int) in the C language, usually 32 bits long, but it can specify a user-defined data types in Vivado HLS, such as using only 16 bit of the data[2].

HLS steps in the process are as follows:

1) Compile, execute (simulate), and debug the C algorithm.
2) Synthesize the C algorithm into an RTL implementation, optionally using user optimization directives.
3) Generate comprehensive reports and analyze the design.
4) Verify the RTL implementation using a pushbutton flow.
5) Package the RTL implementation into a selection of IP formats.

A. *Create Project*

The first step, to compile the project and test different grammatical errors in the design files.

B. *Test Function*

The second step, testing feature of function to be synthesised (in the source file) correct or not. Use C Simulation tool which shown in Fig. 4, where you can also choose to debug, as long as the hook on the Debug function modules. In this step will use test bench to perform this function call, verify that it is functioning properly. modify the design files.

C. *Synthesis*

The third step is to synthesis, Vivado HLS integrated functions defined in the source file. Using C Synthesis tool to generate a number of synthesis reports, including target FPGA clock cycle estimates and latency estimates, as shown in Fig. 5. Generating resource utilization estimation, as shown in Fig. 6. In addition, Vivado HLS also generate loop-related metrics. The main output of this step is a C function of Verilog and VHDL code (RTL design) [9], the interface shown in Fig. 7.

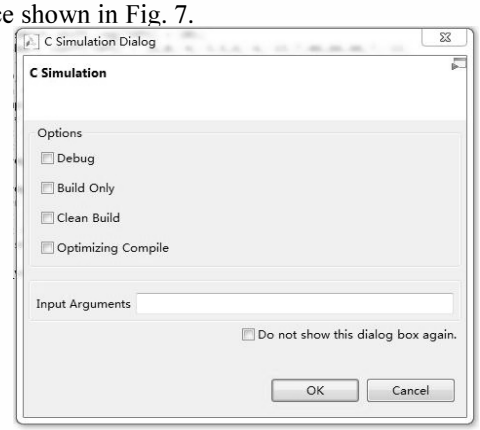

Figure 4. Verify C code

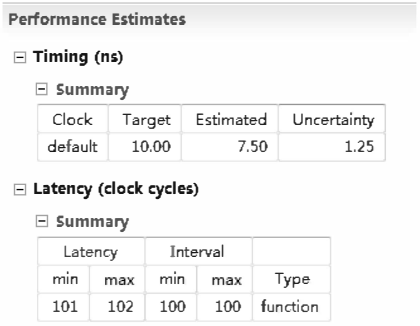

Figure 5. Performance estimates

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	4	0	561
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	14
Register	-	-	537	-
ShiftMemory	-	-	-	-
Total	0	4	537	575
Available	2060	2800	607200	303600
Utilization (%)	0	~0	~0	~0

Figure 6. Utilization estimates

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir_hw	return value
ap_rst	in	1	ap_ctrl_hs	fir_hw	return value
ap_start	in	1	ap_ctrl_hs	fir_hw	return value
ap_done	out	1	ap_ctrl_hs	fir_hw	return value
ap_idle	out	1	ap_ctrl_hs	fir_hw	return value
ap_ready	out	1	ap_ctrl_hs	fir_hw	return value
input_val_V_dout	in	32	ap_fifo	input_val_V	pointer
input_val_V_empty_n	in	1	ap_fifo	input_val_V	pointer
input_val_V_read	out	1	ap_fifo	input_val_V	pointer
output_val_V_din	out	32	ap_fifo	output_val_V	pointer
output_val_V_full_n	in	1	ap_fifo	output_val_V	pointer
output_val_V_write	out	1	ap_fifo	output_val_V	pointer
output_val_V_ap_lwr	out	1	ap_fifo	output_val_V	pointer

Figure 7. Interface

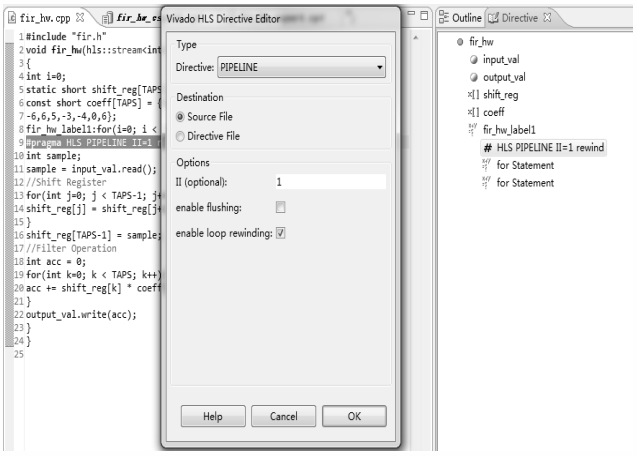

Figure 8. optimization and constraint

There is a very useful feature in the directive, which can set different optimization constraints option for C code, according to the needs of each module. Through the synthesis verification, different optimization constraints were compared. There is a pipeline optimization, as shown in Fig. 8.

High-Level Synthesis seeks to place as many operations from the target device into each clock cycle. The reset style used in the final RTL is controlled, along setting such as the FSM encoding style, using the config_rtl configuration.

The primary optimizations for Optimizing for Throughput are presented together in the manner in which they are typically used: pipeline the tasks to improve performance, improve the data flow between tasks and optimize structures to improve address issues which may limit performance.

Optimizing for Latency uses the techniques of latency constraints and the removal of loop transitions to reduce the number of clock cycles required to complete.

Figure 9. C / RTL co-simulation

D. RTL Co-Simulation

The fourth step is using the C testbench and RTL to do co-simulation. Post-synthesis verification is automated through the C/RTL co-simulation feature which reuses the pre-synthesis C test bench to perform verification on the output RTL. Use Run C / RTL Cosimulation tool, shown in Fig. 9, select the appropriate option depending on the circumstances, it is noted here chose Modelsim. To successfully complete this step, the computer system (Windows or Linux) in the PATH environment variable should contain the ModelSim installation path. This step is called C / RTL co-simulation, because the tool is used testbench C source code for verification before, now using RTL to test the functional correctness.

E. Export RTL / RTL Implement

Finally, the fifth step is to export RTL as IP module, use the Export RTL tool, as shown in Fig. 10. The modules would process by other Xilinx tools for larger designs. RTL can be exported as IP-XACT format IP module and also be exported as IP module System Generator IP module or

pcore format, and then used the Xilinx Embedded Design Suite.

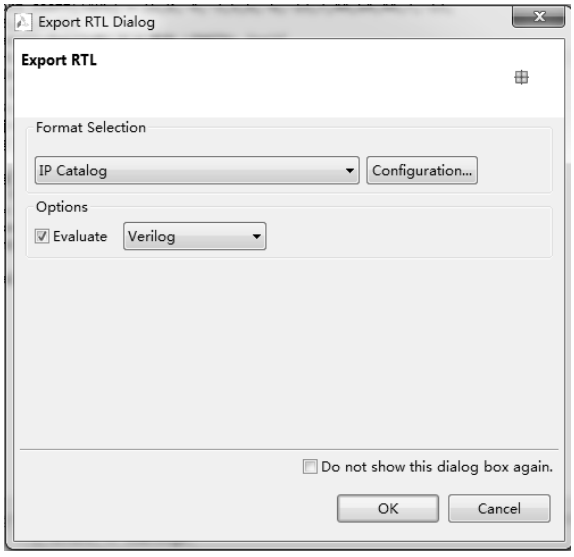

Figure 10. Export RTL

When Vivado HLS reports on the results of synthesis, it provides an estimation of the results expected after RTL synthesis: the expected clock frequency, the expected number of registers, LUTs and block RAMs. These results are estimations because Vivado HLS cannot know what exact optimizations RTL synthesis performs or what the actual routing delays will be, and hence cannot know the final area and timing values.

Before exporting a design, you have the opportunity to execute logic synthesis and confirm the accuracy of the estimates.

When exporting generated RTL, it can select the tool "evaluate" option to evaluate the performance of post-layout and run RTL implementation. The evaluate option shown in Fig. 10 invokes RTL synthesis during the export process and synthesizes the RTL design to gates. In this case, Vivado HLS tool calls Xilinx Vivado tools. To achieve this purpose, the system PATH environment variable should include Vivado installation path.

In addition to the packaged output formats, the RTL files are available as stand-alone files (not part of a packaged format) in the verilog and vhdl directories located within the implementation directory <project_name>/ <solution_name> /impl.

In addition to the RTL files, these directories also contain project files for the Vivado Design Suite. Opening the file project.xpr causes the design (Verilog or VHDL) to be opened in a Vivado project where the design may be analyzed. If co-simulation was executed in the Vivado HLS project, the C/RTL co-simulation files are available inside the Vivado project.

In addition, the generated RTL by Vivado HLS can be used in larger design, or be used as top-level design itself. When large design RTL instantiated exported, should pay attention to the relevant interface requirements.

F. Application

In this paper, a parallel FFT is implemented, it is parallel 8 lines, the points of each line FFT are 2k, thus the total points of FFT are 16k. Here, using the traditional RTL and vivado HLS respectively, compared the implementation of time and resources, as shown in Table I. It can be seen from the table, this paper introduces the vivado HLS can help designers to achieve function quickly while saving resources.

TABLE I. COMPARISON OF TWO METHODS

Parallel FFT	RTL(Verilog)	vivado HLS
Implementation time	2 weeks	1 week
Block ram	109	93
DSP48	266	242

III. CONCLUSION

Designers use Vivado HLS tools can perform various functions in various ways. For example, users can create a design C, C ++ or SystemC expressions, and C testbench used to describe the expected design behavior. Then verify system behavior design with GCC / G ++ or Visual C ++ simulator. Once the behavior is designed to run well, the issue corresponding test platform fully resolved, it can run through Vivado HLS Synthesis design and generate RTL design. This function is very powerful, allowing engineers from different linguistic level to check the design, with the RTL, then can perform Verilog or VHDL design simulation, or create SystemC version uses C wrapper technology tools. Then making System C-level simulation, based on previous C test platform framework created before, verify the design and functionality of the structure behavior. After curing of the design, it can implement the process design kit to run Vivado design, the design programmed into the device, run in hardware or use of IP wrapper to transform the design into reusable IP.

REFERENCES

[1] ug998-Introduction to FPGA Design with Vivado High-Level Synthesis, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/ug871-vivado-high-level-synthesis-tutorial.pdf

[2] ug902-Design Suite User Guide: High-Level Synthesis , http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug902-vivado-high-level-synthesis.pdf.

[3] Jack Zhang, Nine reasons to enhance productivity with Vivado Design Suite, 2013.10

[4] Lina Zhou , VIVADO Improper Use and Advanced , Xilinx, 2014.11

[5] ug871-vivado-high-level-synthesis-tutorial, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/ug871-vivado-high-level-synthesis-tutorial.pdf

[6] wp416-Vivado-Design-Suite, http://www.xilinx.com/support/documentation/white_papers/wp416-Vivado-Design-Suite.pdf

[7] Xianyuan Men, Zhanglin Chen, The new generation of Xilinx FPGA design suite Vivado Application Guide. Tsinghua University Press, 2014

[8] Wenbo Xu, Yun Tian, Xilinx FPGA development of practical tutorials. Tsinghua University Press, 2012

[9] IEEE Behavioural Languages - Part 4: Verilog Hardware Description Language (Adoption of IEEE Std 1364-2001),2004