Finding the best System Design Flow for a High-Speed JPEG Encoder

Kazuo Sakiyama
UCLA Dept of EE
Los Angeles, CA 90095
+1-310-267-4940
kazuo@ee.ucla.edu

Patrick R. Schaumont
UCLA Dept of EE
Los Angeles, CA 90095
+1-310-267-4940
schaum@ee.ucla.edu

Ingrid M. Verbauwhede
UCLA Dept of EE
Los Angeles, CA 90095
+1-310-794-5209
ingrid@ee.ucla.edu

Abstract - 26 students at the University of California, Los Angeles (UCLA) studied system level design methodologies through the design of a high-speed JPEG encoder. The results produced by 5 different design flows onto various target platforms demonstrate the high impact of tools on design quality.

I. Introduction

Growing system complexity and shrinking turn-around -times for silicon chips require the use of efficient design methods and tools. Tool support is needed throughout the design flow, starting with capture of system level models all the way down to detailed implementation. Design decisions at system level have most impact on the final performance, yet the least amount of design support and no textbooks or educational material are available.

Thus the Spring 2002 EE201A class [1], a graduate-level class at UCLA, conducted an experiment to design a high-speed JPEG encoder. Such an encoder is used in image compression applications. A high-speed, energy-efficient implementation is required for embedded applications such as for instance digital cameras. In the class, 5 different approaches were taken to implement the JPEG encoder. We used 3 different design languages (SystemC, HandleC, and SpecC) [2,3]. We targeted 4 different platforms: 2 DSP processors (TI C5410, Analog Devices Blackfin), and 2 dedicated FPGA implementations. One was a VLIW processor created with Adelante Technologies A|RT Designer [4] and the other a dedicated hardware implementation created with Celoxica DK1 [5]. Starting from a single specification in C, we thus could compare a wide variety of design approaches.

II. Overall of Design Flow

The design flow was assigned to the teams as shown in Fig.1. It starts with a single specification and runs through several different design phases. Several tools and environments are used at each level. The numbers next to the arrows indicate the number of teams that followed a particular design trajectory.

At the first step, a data-flow analysis is done of the C code to identify the individual processing stages in the JPEG encoder. In addition we also analyze the background memory requirements of the JPEG encoder in order to optimize the memory architecture of the target platform.

In a second step, each team translates the C reference

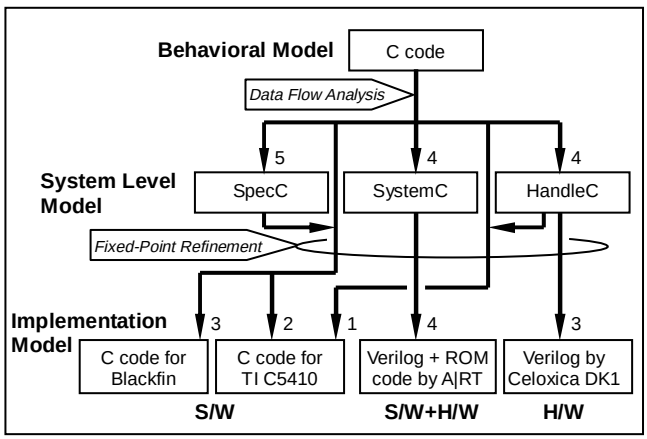

Fig. 1. Design flow for the JPEG encoder. The numbers indicate how many teams were assigned to use the models.

implementation into a system level modeling environment, which is one of SystemC, HandleC, or SpecC.

The third step deals with fixed point refinement. Depending on target platform, appropriate fixed-point refinement has to be done. In case of C5410 or Blackfin, all calculations should fit in 16 bits of precision. In the case of FPGA targets (A|RT Designer or DK1), wordlengths can be custom chosen but should be minimized to reduce resource consumption. For shared busses, it also makes sense to reduce the number of different wordlengths so as to minimize type alignment hardware cost.

The last step is the implementation phase, where each team implements their design on one of 3 reference PCBs: A Spectrum Digital board with C5410 [6], an Analog Devices Blackfin evaluation board [7], and an Insight Electronics board with a Xilinx Virtex-II [8]. The teams that used TI and Analog Devices Blackfin started from the C reference code optimized in the previous steps and compiled the code. A|RT Designer and Celoxica DK1 produce Verilog code, which was implemented with Xilinx ISE software and ported to the FPGA board.

A crucial constraint for all these designs is the design time, which was limited to the time of one quarter (10 weeks).

III. Results

The data flow of the JPEG encoder is based on the block diagram as shown in Fig.2. It has 5 functional units and 7 memory units. The functional units implement the different

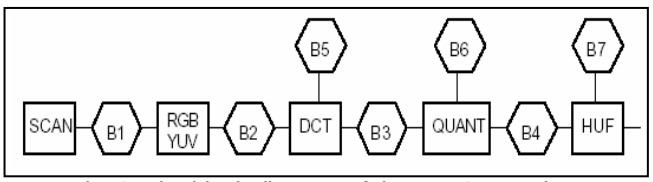

Fig. 2. The block diagram of the JPEG encoder.

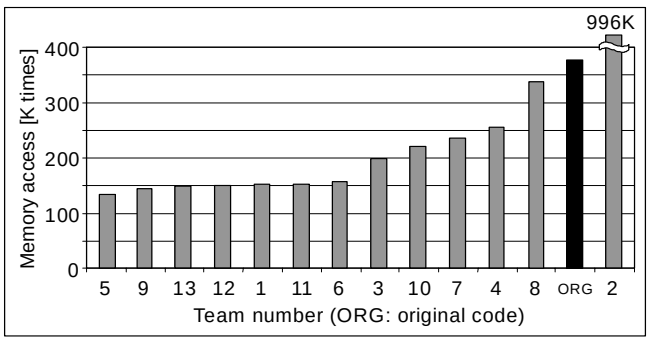

Fig. 3. Optimization of the memory accesses for each team.

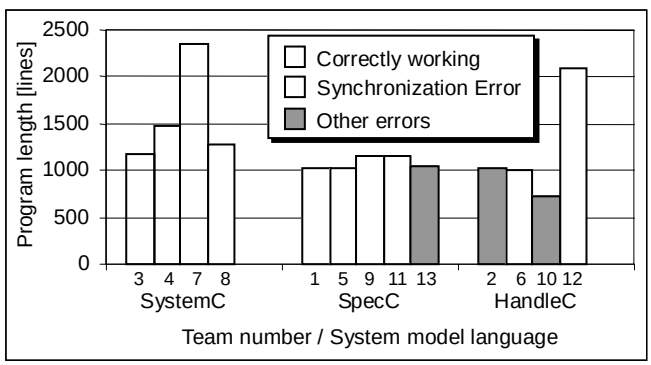

Fig. 4. The program length for each system model language and the correctness of the code.

steps of JPEG encoding, including reading an image, RGB to YUV color space conversion, DCT coding, quantization and Huffman encoding.

A. The Number of Memory Access

Fig.3 shows the number of memory accesses optimized by each team using ATOMIUM [9], a tool that evaluates the read and write access count in the C code. The goal of this step is to reduce the number of memory accesses, and in turn this enables more efficient use of storage. More than half of the teams were able to reduce the access count by 50% or more.

B. System Level Model

The number of lines of code programmed with each system language is shown in Fig.4. The design time for this step was two weeks. The result with SpecC says that 4 teams out of 5 wrote about the same amount of program working correctly. On the contrary, the code did not work successfully by the due date with 3 SystemC teams and 2 HandleC teams out of 4 each, and varied enormously in programming style among the teams. We attribute this variation in results mostly on the lack of a clear well-documented design approach at the time of the project.

Table I
The JPEG encode performance for each platform

Platform (source code)	Average of Required Cycles [cycles/64blocks]	Code Length [lines]	JPEG encode Performance [blocks/s]
Blackfin (C code)	1,524K	879	12,602 @300MHz
TI C5410 (C code)	1,499K	707	6,835 @160MHz
A|RT (SystemC)	677K	1,015	-
DK1 (HandleC)	700K	1,312	1,357@15MHz (Simulation Value)

C. Implementation on the reference board

After the fixed-point refinement suitable for each platform, every team tried to run the JPEG encoder on the reference boards. All of the teams using DSP platform could implement their design successfully onto the boards. However, no team targeting the FPGA board could obtain an implementation within the 10 weeks time limit.

Based on the data reported by the teams, the JPEG encode performance is calculated and compared in Table I.

IV. Summary and Conclusions

We draw a fourfold conclusion from this design experiment. First, high-level design greatly benefits from advanced optimizations such as reduction of memory accesses. Next, the use of a system level design language requires a well-documented design methodology to be effective. Third, current FPGA-based design flows do not yet have the same ease of use as equivalent DSP-based design flows. Finally, gaps in the design flow have been a constant source of extra effort during the entire project. These gaps cause overhead because they require rewrite of code, such as for instance to go from SpecC to a DSP. A design flow must be closed to be fully effective.

Acknowledgements

We would like to acknowledge all of the graduate students who enrolled EE201A in Spring Quarter at UCLA. We also acknowledge the logistical contributions from Analog Devices, Adelante Technologies, and Celoxica.

References

[1] http://www.ee.ucla.edu/~schaum/ee201a/
[2] http://www.systemc.org/
[3] http://www.ics.uci.edu/~specc/
[4] http://www.adelantetech.com/en/html/algemeen/AboutAdelante/Partners/Academic/EducationalProgram.asp
[5] http://downloads.celoxica.com/dk1eval/
[6] http://focus.ti.com/docs/tool/list.jhtml?familyId=114&toolTypeId=30
[7] http://www.analog.com/technology/dsp/products/ads/blackfin/index.html
[8] http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Platform+FPGAs
[9] http://www.imec.be/design/multimedia/atomium/