An Independent Evaluation of:

High-Level Synthesis Tools for Xilinx FPGAs

By the staff of

Berkeley Design Technology, Inc.

Executive Summary

In 2009, Berkeley Design Technology Inc. (BDTI), an independent benchmarking and analysis firm, launched the BDTI High-Level Synthesis Tool Certification Program™ to evaluate high-level synthesis tools for FPGAs. Such tools take as their input a high-level representation of an application (written in C or MATLAB, for example) and generate a register-transfer-level (RTL) implementation for an FPGA. Thus far, two high-level synthesis tools, **AutoESL's AutoPilot** and the **Synopsys Synphony C Compiler**, have been certified under the program.

BDTI's evaluation program uses two example applications, a video motion analysis application and a wireless receiver, to evaluate high-level synthesis tools (HLSTs) on a number of quantitative and qualitative metrics. As shown in Figure 1 and Figure 2, we found that the Xilinx Spartan-3A DSP 3400 FPGA used with either of the two HLSTs provided roughly 40X better performance than a mainstream DSP processor, and that the high-level synthesis tools were able to achieve FPGA resource utilization levels comparable to hand-written RTL code. Furthermore, as we will discuss in this white paper, implementing our video application using the HLSTs along with Xilinx FPGA tools required a similar level of effort as that required for the DSP processor. This finding will no doubt be surprising to many, as FPGAs have historically required much more development time than DSPs.

Based on our analysis, we believe that HLSTs can significantly increase the productivity of current FPGA users. For those using DSP processors in highly demanding applications, we believe that FPGAs used with HLSTs are worthy of serious consideration.

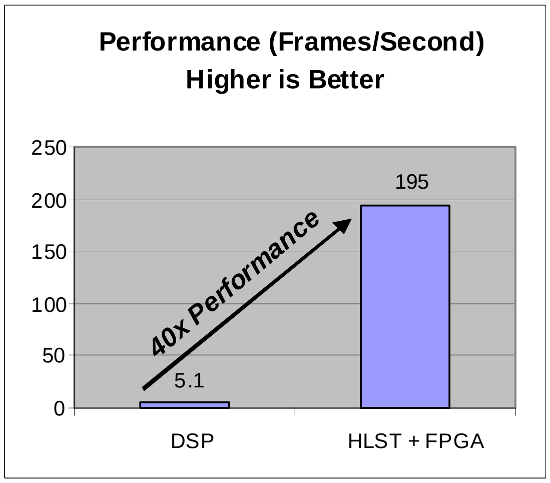


FIGURE 1. **Maximum frame rate achieved (at 720p resolution) on the BDTI Optical Flow Workload (a video application) on a DSP processor and a Spartan-3A DSP FPGA using HLSTs.**

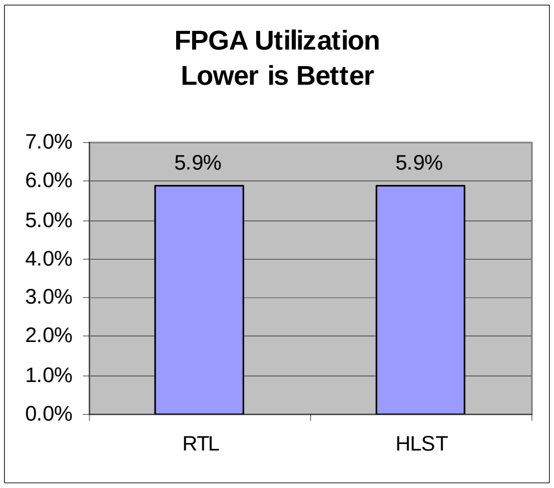


FIGURE 2. **FPGA resource utilization on the BDTI DQPSK Workload (a wireless receiver) implemented using HLSTs (average result) versus hand-written RTL code.**

In this paper, we describe the methodology underpinning our evaluation, present selected results, and highlight key advantages and limitations of high-level synthesis tools as used with Xilinx FPGAs.

Background

Xilinx was founded in 1984 and has long been a dominant provider of FPGAs. Current Xilinx FPGA families include the high-performance Virtex-5 and Virtex-6 families, and the low-cost Spartan-3 and Spartan-6 families. The company has been active in developing new FPGA chips and tools, and in recent years has focused on creating application-domain-specific variants of its chips and tools. In particular, Xilinx has developed a suite of "Targeted Design Platforms" that are intended to help users accelerate development in specific application areas. For example, the video application used in this evaluation was implemented on a Spartan-3A DSP FPGA using the XtremeDSP Video Starter Kit — Spartan-3A DSP Edition, which is a Targeted Design Platform.

The shift to using FPGAs as processing engines is a relatively new phenomenon. When FPGAs first became commercially available, they lacked sufficient capacity to be used as processing engines. Instead, they were used for "glue logic" to facilitate interfacing among other chips. But as their capacity has grown and their architectures have incorporated specialized features such as multipliers and distributed memories, FPGAs have increasingly found use as powerful parallel processing engines. In 2007 BDTI published a ground-breaking report, *FPGAs for DSP*, that included benchmark results showing that FPGAs could achieve 100X higher performance and 30X better cost-performance than DSP processors in some highly parallelizable signal processing applications—a dramatic advantage. Nevertheless, the widespread adoption of FPGAs as processing engines has been hampered by the challenges of using FPGAs. Implementing an application in hand-coded RTL HDL (hardware description language) code is typically far more labor intensive and error prone than implementing the application on a programmable processor, and requires a very different skill set.

High-level synthesis tools take as their input a high-level representation of desired functionality and generate an RTL HDL description of a hardware implementation targeting an FPGA or ASIC. HLSTs can easily extract parallel circuit implementations from loops that have a large number of operations with limited data dependencies, and allow incorporation of concurrency into a design while still working with a familiar high-level language. Thus, high-level synthesis tools eliminate the step of manually creating the RTL implementation. They can also eliminate much of the effort in developing the test benches needed to verify the RTL implementation. Furthermore, much of the debugging and verification can be performed at a high level rather than at the RTL code level, which can significantly reduce debugging and verification time and effort.

For processor users seeking higher performance, a key question is whether high-level synthesis tools can make FPGA design more like processor software development, in terms of productivity and skills requirements. For current FPGA designers, the key question is whether HLSTs deliver substantial improvements in productivity without compromising the performance and efficiency of the design.

The idea of high-level synthesis is often met with skepticism. This is largely due to the common belief that software tools cannot produce results as good as those produced by a skilled engineer. Unfortunately, there are plenty of examples of older high-level synthesis tools—not to mention older software compilers for processors—that support this viewpoint.

However, there is a growing body of anecdotal evidence suggesting that some modern high-level synthesis tools are very effective, both in terms of usability and quality of results. (See [1] for an excellent tutorial on the history and evolution of high-level synthesis tools.) Given this conflicting information, how is a prospective user to judge whether high-level synthesis tools are worth considering?

BDTI created the BDTI High-Level Synthesis Tool Certification Program to provide objective, credible data and analysis to enable potential users of high-level synthesis tools for FPGAs to quickly understand the capabilities and limitations of these tools.

This white paper explains the evaluation methodology developed for this program and provides an overview of the results obtained from BDTI's in-depth, hands-on evaluation of two high-level synthesis tools—AutoPilot from AutoESL, and the Synphony C Compiler from Synopsys—used with a Xilinx FPGA and with Xilinx's RTL tools. In this paper, we present representative results for Xilinx FPGAs used in conjunction with the two HLSTs evaluated thus far. Detailed results for these tools are available on BDTI's website, www.BDTI.com.

About BDTI

Berkeley Design Technology, Inc. (BDTI) is an independent technology analysis, consulting, and engineering services company headquartered in Oakland, California. Founded in 1991, BDTI is widely known for its highly regarded, independent performance analysis of processing platforms for embedded applications. BDTI's six suites of chip benchmarks have been licensed for use with nearly 100 processing platforms, from MCUs to FPGAs.

Unlike market research firms, BDTI is made up of engineers with real-world design experience; the company's analysis projects often involve extensive hands-on work with chips, tools, and software. Engineers at BDTI have significant expertise in algorithm and software development for a range of processors, but are not experienced in FPGA design. As such, they approached this project as FPGA novices, and are representative of other DSP processor users who may be considering a switch to FPGAs.

Further information about the company along with a variety of analysis results are available at www.BDTI.com.

Design Using High-Level Synthesis Tools

In this evaluation program, applications are implemented on a Xilinx FPGA using the high-level synthesis tools via two main steps: First, starting with a high-level language description of the desired functionality, the high-level synthesis tool is used to generate an RTL implementation. Then, Xilinx's RTL tools (the Integrated Synthesis Environment, or ISE, and the Embedded Development Kit, or EDK) are used to transform that RTL implementation into a complete FPGA implementation in the form of a bitstream for programming a specific FPGA on a specific hardware platform with I/O and memory. (In our case, the platform is the Video Starter Kit mentioned earlier.) The high-level synthesis tools also generate a wrapper for the RTL implementation so that the result can be used as a core in the EDK tools in combination with I/O and external memory. In this paper we use the term "RTL tools" to refer to the combination of the ISE and EDK tools.

One could limit an evaluation of high-level synthesis tools to the use of the synthesis tools alone, ignoring the RTL-to-bitstream portion of the design flow. But potential users need to know how difficult (or easy) it is to get from the high-level application description all the way to an FPGA implementation—which requires the RTL tools in addition to the high-level synthesis tool. For this reason, BDTI evaluates the *entire* implementation process, that is, not just the C-to-RTL portion, but also the Xilinx RTL tool chain. The full implementation process is shown in Figure 3.

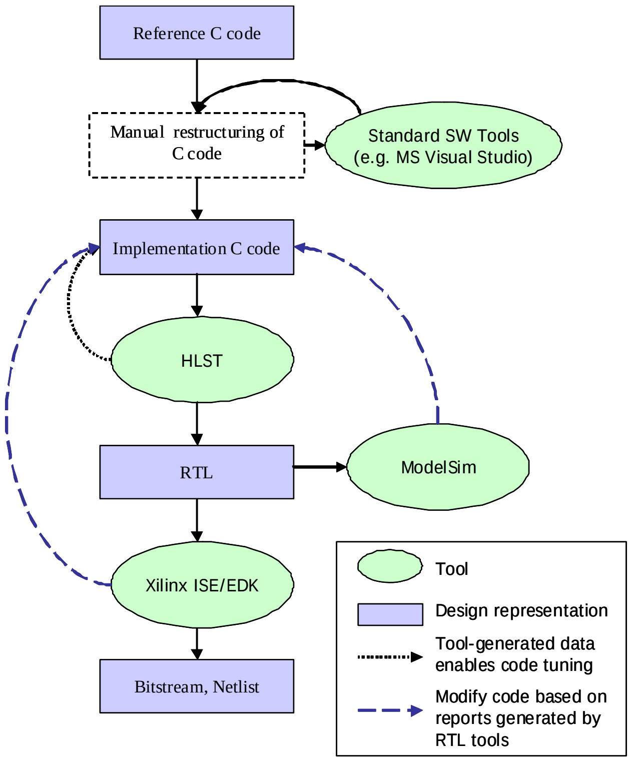

Reference C code
Manual restructuring of C code
Standard SW Tools (e.g. MS Visual Studio)
Implementation C code
HLST
RTL
ModelSim
Xilinx ISE/EDK
Bitstream, Netlist
Tool
Design representation
Tool-generated data enables code tuning
Modify code based on reports generated by RTL tools

FIGURE 3. Design flow using the AutoPilot or Synphony C Compiler high-level synthesis tools in conjunction with Xilinx RTL tools. (Figure based on a diagram provided by AutoESL.)

As illustrated in Figure 3, the first step in implementing an application on any hardware target is often to restructure the initial C code. By "restructuring," we mean rewriting the initial C code (which is typically coded for clarity and ease of conceptual understanding rather than for optimized performance) into a format more suitable for the target processing engine. This typically involves modifying the code such that the size of data structures is correlated with the size of on-chip memories and unnecessary data dependencies are removed, among other changes.

On a DSP processor, for example, it may be appropriate to rearrange an application's control flow so that intermediate data always fits in cache. On an FPGA with high-level synthesis tools, restructuring typically provides a representation of an application that allows the HLS tools to extract potential parallelism, resulting in a streaming pipelined implementation.

Restructuring can sometimes enable improvements of several orders of magnitude in speed or resource utilization. If the original C code is not appropriate for the hardware target, restructuring may be required in order to get the design to work in real time or to fit within the available hardware resources. For example, a video application may pass frames of intermediate video data between algorithm blocks. A straightforward C representation of the application may require large frame buffers that would not fit in an FPGA, but the code can typically be restructured to enable use of much smaller buffers. By appropriately streaming the data flow between algorithm kernels, the application can buffer just a few scan lines—or even just a few pixels—rather than buffering entire frames. In some cases, applications may require the use of external memory (for example, if a video application depends on values from previous frames, it must store entire frames, which may not fit on the chip). In this case, the restructuring process requires the addition of code and interfaces that support the use of an external memory interface. The effort required for restructuring can vary significantly, depending on the application, the required performance, and the target platform.

In general, high-level synthesis tools do not handle restructuring automatically. Instead, the restructuring is done by hand. In fact, initial restructuring work tends to be unrelated to the specifics of the high-level synthesis tool, and can be done independently of the tool. (In our evaluation, for example, we used Microsoft Visual Studio for initial restructuring and verification of the C code.) The high-level synthesis tool is then used to generate an initial implementation, and further restructuring may then be done based on the performance and resource use of the initial implementation. Compared with hand-written RTL code, where restructuring and language translation are performed as a single combined step, restructuring entirely in C is easier and less error-prone—an advantage for high-level synthesis tools.

The C code may also need to be modified to take advantage of the FPGA's ability to implement arbitrary-precision data paths and other features. To facilitate this, high-level synthesis tools typically support arbitrary-width data types.

In addition to the obvious potential productivity gains associated with writing and manipulating a high-level functionality description rather than a low-level implementation, high-level synthesis tools offer significant advantages for verification. For example, some high-level synthesis tools generate high-level, SystemC representations of synthesized designs, enabling much faster simulation runs compared with traditional RTL simulation. However, high-level simulations may not simulate all of the interfaces of the synthesized hardware (such as I/O and memory subsystems), so more detailed simulation may still be needed for full verification of the full, integrated system. Additionally, many users run low-level simulations on a subset of their test data to sanity-check the high-level simulations. To facilitate SystemC and RTL simulations, high-level synthesis tools typically generate the needed test benches based on high-level test benches provided by the user, as shown in Figure 4 below.

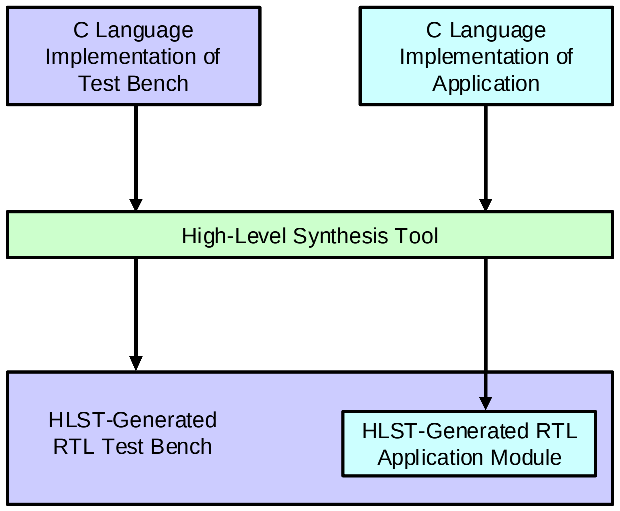

C Language Implementation of Test Bench
C Language Implementation of Application
High-Level Synthesis Tool
HLST-Generated RTL Test Bench
HLST-Generated RTL Application Module

FIGURE 4. Typical HLSTs automatically generate RTL test benches in addition to the RTL module implementing the design.

After restructuring the high-level code, the user directs the HLST to synthesize a hardware implementation of the specified functionality in the form of RTL HDL code. HLS tools can provide an estimate of the resource utilization and clock frequency that the synthesized implementation is likely to meet. Obtaining this estimate does not require invoking the RTL tools and can be obtained at any point in the design process, enabling early design exploration, for example to identify achievable cost-performance points. In our evaluation, the HLS tool resource usage estimates for flip-flops, look-up-tables, and DSP blocks were accurate (within 10%), while estimates for block RAM were consistently less than the actual usage (on the order of 25% less than actual). The user can utilize the performance and resource-use estimates generated by the HLST to assess the synthesized implementation and, as necessary, make improvements to the implementation by modifying the high-level representation.

At this point the process switches over to more traditional RTL tools. In our evaluation, Xilinx's RTL tools (ISE and EDK) were used to take the RTL code generated by the HLST, perform synthesis and place-and-route tasks, report the exact resource utilization of the implementation, and alert the user to any timing issues. The user can then further tune the C source code and repeat the RTL implementation generation if needed.

This is a particular area of weakness for the HLST + Xilinx tool flow. Limited integration between the HLSTs and Xilinx RTL tools can make it challenging to resolve design problems that are uncovered outside the scope of the HLS tools. In an ideal world the HLS tools would perfectly model the FPGA and such problems would never be encountered, or at least there would be a tighter coupling between HLSTs and the RTL tools to streamline the debugging process. In reality, however, our evaluation showed that if the HLST user does not have RTL design and tools skills, the assistance of an engineer with those skills will be needed at this stage of the design flow.

The BDTI High-Level Synthesis Tool Certification Program

BDTI evaluates high-level synthesis tool flows (including the associated RTL tools) using two well-defined sample applications, or "workloads." These applications (described briefly in the next section) are representative of demanding digital signal processing applications, typically requiring the performance associated with an FPGA. The two applications are implemented using several approaches. First, a given workload is implemented on the target FPGA using the high-level synthesis tool in conjunction with the Xilinx RTL tools. The same workload is then implemented on the same FPGA using a traditional RTL design approach, or on a DSP processor using its associated development tools (depending on the workload under consideration). In this manner, BDTI is able to compare the quality of results and productivity associated with using various tools-plus-chip combinations.

The two workloads have been chosen to be broadly representative of the types of embedded computing applications that electronic system designers implement using FPGAs. They have high data rates and high computational loads and as such, they are inherently well suited for FPGAs. This is an important point to keep in mind. There are many important applications (such as high-definition audio codecs) that don't require the computational performance levels or data rates of the workloads used here, and that require much more complex algorithms. Such applications may yield different results (in terms of ease of use, productivity, performance, or resource utilization) than those reported here.

Evaluation Workloads

The two applications used in BDTI's evaluation are the BDTI Optical Flow Workload™ and the BDTI DQPSK Receiver Workload™.

The term "optical flow" (or "optic flow") refers to a class of video processing algorithms that analyze the motion of objects and object features (such as edges) within a scene. The BDTI Optical Flow Workload operates on a 720p resolution (1280×720 progressive scan) input video sequence and produces a series of two-dimensional matrices characterizing the apparent vertical and horizontal motion within the sequence. In designing this workload, BDTI has increased the control complexity relative to what might be found in a simple optical flow algorithm, in order to create a workload that is more representative of a complete real-world application and that represents a challenging test case for the tools. More specifically, BDTI incorporated dynamic, data-dependent decision making and array indexing into the optical flow workload.

There are two Operating Points associated with the BDTI Optical Flow Workload, each of which uses the same algorithm but is optimized for a different metric.

- Operating Point 1 is a fixed workload defined as processing video with 720p resolution at 60 frames per second. The objective for Operating Point 1 is to *minimize the resource utilization* required to achieve the specified throughput. (Resource utilization refers to the fraction of available processing engine resources required to implement the workload.)
- Operating Point 2 is defined as the maximum throughput capability of the workload implementation on the target device for 720p resolution. The objective for Operating Point 2 is to *maximize the throughput* (measured in frames per second) using all available device resources.

The second workload is the BDTI DQPSK Receiver Workload. This workload is a wireless communications receiver baseband application that includes classical communications blocks found in many types of wireless receivers. It is a fixed workload with a single Operating Point defined as processing an input stream of complex, modulated data at 18.75 Msamples/second with the receiver chain clocked at 75 MHz. The receiver produces a demodulated output bit stream of 4.6875 Mbits/second. The objective for this workload is to *minimize the FPGA resource utilization* needed to achieve the specified throughput.

Memory usage and memory bandwidth requirements vary significantly among the workloads. The BDTI DSPSK Receiver Workload requires minimal memory usage (and therefore no external memory chip). The BDTI Optical Flow Workload, however, requires storing a history of four video frames (1280×720 pixels per frame), and thus requires an external memory chip to accompany the Spartan-3A DSP FPGA. Optical Flow Workload Operating Point 1 requires a single external memory chip and interface (with a bandwidth of approximately 450 Mbytes/second), while Optical Flow Workload Operating Point 2 typically requires two external memory chips and interfaces (with a combined bandwidth of approximately 1.4 Gbytes/second).

For the BDTI Optical Flow Workload, typical FPGA implementations process one pixel per clock cycle for Operating Point 1 and two pixels per clock cycle for Operating Point 2. BDTI DQPSK Receiver Workload implementations process one input sample every four clock cycles.

Description of Metrics

Historically, demanding applications that were implemented in hand-written RTL code on an FPGA typically provided good quality of results (in terms of performance and efficiency) but poor productivity, while applications implemented on DSP processors provided good productivity but relatively poor quality of results. High-level synthesis tools targeting FPGAs seek to provide the best of both worlds: good quality of results achieved with high productivity levels. Therefore, in our evaluation we consider two categories of metrics:

- **Quality of results metrics** assess the performance and resource utilization of the workload implementation. For the BDTI Optical Flow Workload, quality of results metrics are reported for the HLST-Xilinx flow and for the DSP processor flow. For the BDTI DQPSK Receiver Workload, quality of results metrics are reported for the HLST-Xilinx flow and for a traditional FPGA implementation using a hand-written RTL design developed in accordance with typical industry design practices, including the use of Xilinx Coregen IP blocks where appropriate.
- **Usability metrics** assess the productivity and ease of use associated with the HLST-Xilinx design flow, and are based on the BDTI Optical Flow Workload. These metrics compare the productivity and ease of use associated with using the HLST and Xilinx tools targeting an FPGA relative to using a DSP processor with its associated software development tool chain.

Usability metrics are evaluated qualitatively based on nine aspects of tool use, including out-of-the-box experience, ease of use, completeness of tool capabilities, efficiency of overall design methodology, and quality of documentation and support.

Description of Platforms

For this evaluation, the target FPGA is the Xilinx Spartan-3A DSP 3400 (XC3SD3400A). For the BDTI Optical Flow Workload, the Xilinx XtremeDSP Video Starter Kit — Spartan-3A DSP Edition is used as the target platform. Spartan-3A DSP FPGAs are based on Xilinx's low-cost Spartan-3A family, but add a number of enhancements to accelerate digital signal processing. For example, Spartan-3A DSP chips have double the block RAM of other Spartan devices and incorporate hard-wired DSP data paths, called "DSP48A slices." Each DSP48A slice contains an 18×18 multiplier with pre-adders and an accumulator, among other features. The XC3SD3400A includes 126 DSP48A slices that can be clocked at up to 250 MHz, and roughly 54,000 logic cells. (In early 2009 Xilinx announced its next-generation low-cost FPGA family, the Spartan-6 family. BDTI has not yet evaluated this family.)

Xilinx RTL tools, including the ISE and EDK tool suites, were used with the high-level synthesis tools. (We used ISE and EDK version 10.1.03 (lin64)).

The target DSP processor for this project is the Texas Instruments TMS320DM6437. The TMS320DM6437 is a video-oriented processor that includes a 600 MHz Texas Instruments TMS320C64x+ DSP core along with video hardware accelerators. (The hardware accelerators are not applicable to the BDTI Optical Flow Workload, and therefore were not used.) The evaluation used the Texas Instruments DM6437 Digital Video Development Environment as the target platform, and used the Texas Instruments Code Composer Studio tools suite (version V3.3.82.13, Code Generation Tools version 6.1.9).

We should note here that high-level synthesis tools cost considerably more than DSP processor software development tools. Tool cost is not reflected in the cost-performance results presented later in this paper. A fundamental question is whether the higher tool cost is justified by higher quality of results or higher productivity enabled by these tools. We believe that the results and analysis presented in this paper will prove valuable to prospective users in answering that question.

Implementation, Certification Process

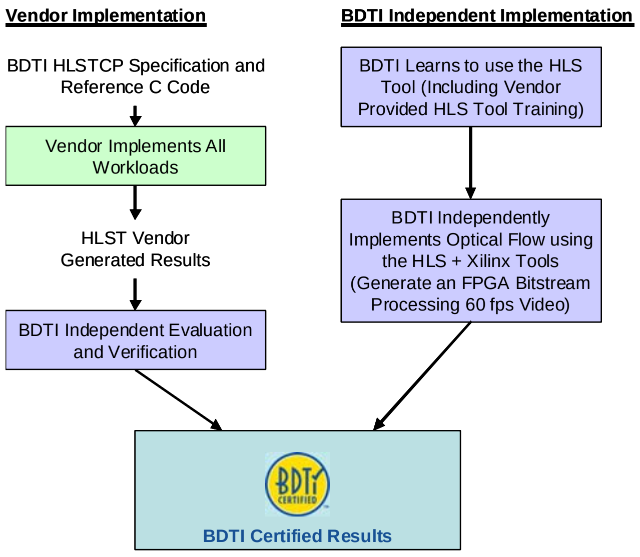
The evaluation process is illustrated in Figure 5. The work of implementing the two workloads on the two chips was distributed between the high-level synthesis tool vendors, Xilinx, and BDTI based on the chip and tool chain used. The high-level synthesis tool vendors implemented both workloads using their tools along with the Xilinx tools and submitted performance and resource utilization results to BDTI for verification and certification. These certified results were used to generate the quality of results metrics presented in this paper.

In parallel, BDTI's engineers received training from the HLS tools vendors and then independently implemented portions of the BDTI Optical Flow Workload using the high-level synthesis tools and Xilinx tools. This process provided BDTI with first-hand insight into the usability of the tool chains and the quality of results they generated. BDTI also implemented the BDTI Optical Flow Workload on the DSP processor, while Xilinx implemented the hand-written RTL FPGA version of the BDTI DQPSK Receiver Workload (which was then verified and certified by BDTI). The DQPSK Receiver RTL design was constructed using typical industry design practices, including the use of two Xilinx Coregen IP blocks.

In addition, BDTI interviewed a number of high-level synthesis tool users about their experiences in using the tools and the results they have attained. These interviews were used to augment (and sanity-check) the results obtained using the workload implementations.

Quality of Results: Performance and Efficiency

The results presented in Figure 1 in the executive summary showed that for our video application, the FPGA implementations created using high-level synthesis tools achieved roughly 40X the performance of the



Vendor Implementation
BDTI Independent Implementation
BDTI HLSTCP Specification and Reference C Code
Vendor Implements All Workloads
HLST Vendor Generated Results
BDTI Independent Evaluation and Verification
BDTI Learns to use the HLS Tool (Including Vendor Provided HLS Tool Training)
BDTI Independently Implements Optical Flow using the HLS + Xilinx Tools (Generate an FPGA Bitstream Processing 60 fps Video)
BDTI Certified Results

FIGURE 5. Process for BDTI's High-Level Synthesis Tool Certification Program.

DSP processor implementation. Bringing chip cost into the analysis, Figure 6 shows the corresponding cost/performance advantage, which is roughly 30X in favor of the FPGA implementations.

Clearly, FPGAs used with high-level synthesis tools can provide a compelling performance and cost-performance advantages for some types of applications.

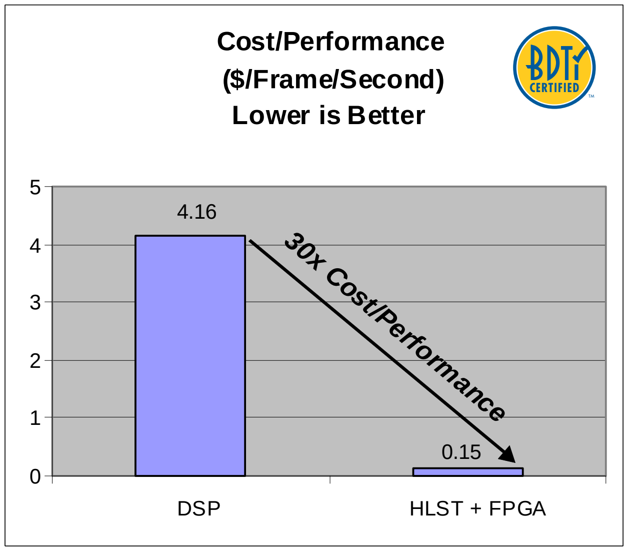


FIGURE 6. Cost/Performance for the BDTI Optical Flow Workload (720p) on a 600 MHz TI 'C64x+-based DSP versus Spartan-3A DSP FPGA using HLSTs.

BDTI also evaluated the efficiency of the HLST-based FPGA implementations of the DQPSK workload versus the same workload implemented using hand-coded RTL. Here, too, the HLSTs performed extremely well. As shown in Table 1, both AutoPilot and Synphony C Compiler were able to generate RTL code that was comparable in efficiency (i.e., resource usage) to that of hand-written RTL code. The similarity of HLS tool and hand-written RTL results is probably not coincidental; AutoESL and Synopsys were provided with the resource utilization figure for the hand-coded RTL implementation at the outset of the evaluation process, and likely used this as a target in optimizing their implementations. (We should note, however, that such information is not required for effective use of the HLS tools, and that the HLS tool vendors were not provided with the hand-written RTL design.)

Interviews with users who have worked with the AutoESL and Synopsys high-level synthesis tools confirmed this resource usage finding: users reported that the tools produce excellent results that are comparable to hand-written RTL code, with much less design and verification effort—a significant achievement.

TABLE 1. DQPSK Receiver Workload. Fixed Throughput (18.75 Msamples/Second Input Data with a 75 MHz Clock Speed)

Platform	Chip Resource Utilization (Lower is Better)
HLST plus Xilinx RTL tools targeting the Xilinx XC3SD3400A FPGA	5.6 - 6.4%
Hand-written RTL code using Xilinx RTL tools targeting the Xilinx XC3SD3400A FPGA	5.9%

Usability Metrics

BDTI's usability metrics provide an assessment of how easy it is to use the high-level synthesis tool flow versus using the DSP processor tool chain. For each usability metric, BDTI assigns a score of Excellent, Very Good, Good, Fair, or Poor. In assigning these scores, BDTI considers the overall design methodology for a complete project—starting with a C language application specification and ending with a real-time implementation on the target chip (either an FPGA or DSP processor).

As shown in Table 2 and Table 3, the scores for the overall HLS tool flow are quite similar to those for the TI DSP processor. Note that the overall HLS tool flow score combines the scores for the HLS tools and the Xilinx RTL tools, and in some cases (such as the Out-of-Box Experience and Ease of Use) the HLS and Xilinx scores were quite different, with the HLS tools scoring significantly higher than the Xilinx tools. In general, the high-level synthesis tools we evaluated were quite straightforward to install and use, even without expertise in FPGA design. In contrast, we had difficulty installing and using the FPGA RTL tools. Therefore, BDTI brought in an experienced FPGA engineer to assist with this aspect of the project. The FPGA engineer's skills were required to interpret messages from the Xilinx RTL tools and interface the HLST-generated RTL modules with I/O and memory modules to create a complete design that runs on the FPGA. In the case of the BDTI Optical Flow Workload, the effort required to develop the I/O and memory interfaces was mitigated because BDTI was able to use many of the platform components provided with the Video Starter Kit. In other system designs, tasks involving I/O and external memory interfaces may require a platform development phase.

BDTI's scores for the usability metrics for the HLS tool flow are assigned based on the following usage model: a "hardware-aware" DSP software engineer handling the aspects of the design involving the high-level synthesis tool, and an experienced FPGA engineer handling the aspects involving the RTL tools. (By "hardware-aware," we mean a DSP software engineer comfortable with basic hardware concepts such as pipelining and DMA.)

Although the HLS tools do not completely abstract the user from the underlying Xilinx RTL tools, the user is partially shielded from them. For example, no manual RTL code was written to implement the BDTI Optical Flow Workload algorithmic processing (this RTL code was generated by the HLSTs) but a few lines of RTL code were written by hand to integrate the algorithmic processing block with other required system components.

The net result, as indicated by the Efficiency of Design Methodology scores, was that the overall HLST-Xilinx tool chains yielded similar productivity to the DSP processor flow. That is, it took a similar level of effort to implement the BDTI Optical Flow Workload on the TI DSP processor as on the Xilinx FPGA using either of the two HLSTs. This is a significant finding; development time has been a key impediment for many system designers trading off the use of a programmable DSP processor versus an FPGA, and our evaluation indicates that this impediment is largely eliminated for applications such as the BDTI Optical Flow Workload.

Furthermore, although the BDTI Optical Flow Workload implementations developed using the HLS tools and the DSP tools both required modifications to the reference C code to obtain efficient implementations, the extent of modifications was less for the HLS tool flows than for the DSP tool flow, as shown by the Extent of Modifications Required column in Table 3.

As shown in Table 4, two different skill sets are required for the use of the high-level synthesis flows: skills related to use of the high-level synthesis tool, and skills related to using the RTL tools. For optimized DSP processor software development, specialized programming skills are required. For example, DSP software engineers typically use optimization techniques like software pipelining and single-instruction, multiple-data operations, and have knowledge of hardware concepts like pipelining, latency, and DMA. Knowledge of these hardware concepts is also helpful in developing an FPGA implementation using a HLS tool. In our experience, a hardware-aware DSP software engineer can learn to effectively use a high-level synthesis tool.Typically a learning curve on the order of several weeks to a few months is required to become proficient (depending on the background of the engineer and the complexity of the design).

TABLE 2. Usability metrics (1 of 2)

	Out-of-Box Experience	Ease of Use	Completeness of Capabilities	Quality of Documentation and Support
Combined HLST + Xilinx RTL tools rating[a]	Fair	Good	Good	Good
Texas Instruments software development tools rating[b]	Good	Very Good	Very Good	Very Good

a.HLS tools plus Xilinx RTL tools targeting a Xilinx XC3SD3400A FPGA
b.Texas Instruments software development tools targeting a TMS320DM6437 DSP processor

TABLE 3. Usability metrics (2 of 2)

	Efficiency of Design Methodology				Extent of Modifications Required to Reference Code
	Learning to Use the Tool	Design and Implementation (First Compiling Version)	Design and Implementation (Final Optimized Version)	Platform Infrastructure Development	
Combined HLST + Xilinx RTL tools rating[a]	Very Good	Very Good	Good	Good	Good
Texas Instruments software development tools rating[b]	N/A (assuming already familiar)	Excellent	Good	Good	Fair

a.HLS tools plus Xilinx RTL tools targeting a Xilinx XC3SD3400A FPGA
b.Texas Instruments software development tools targeting a TMS320DM6437 DSP processor

While we found the Synopsys Synphony C Compiler and AutoESL AutoPilot high-level synthesis tools reasonably easy to use, it is important to note that high-level synthesis tools do not provide a sufficiently complete abstraction to enable an FPGA designer to work exclusively at a high level. Though we found it relatively straightforward to get good RTL results from the high-level synthesis tools, the remaining task of getting from RTL code to bitstream using the Xilinx RTL tool chain was less straightforward. It is not something that most DSP software engineers would be equipped to handle without assistance from an FPGA expert or spending time to become familiar with the Xilinx RTL tools and RTL design concepts. In summary, it's clear that, while the two high-level synthesis tools we have evaluated can produce efficient results and improve productivity, they do not eliminate the requirement for an engineer with FPGA skills to be part of the team—and for current FPGA users, high-level synthesis tools accelerate a significant portion of the development flow process but not the entire flow.

Conclusions

BDTI's earlier benchmarking of FPGAs and DSP processors showed large performance and cost-performance advantages for FPGAs on some applications when the FPGA implementations were created using traditional RTL design techniques. The new analysis presented here confirms this performance advantage (e.g., a 30X cost-performance advantage on the BDTI Optical Flow Workload) and shows that FPGAs can achieve similar performance and cost-performance advantages when

TABLE 4. BDTI High-Level Synthesis Tool Certification Program Results Skills Required

Tool	Required Skill Set
High-level synthesis tools	• Application expertise • C programming • Hardware architecture fundamentals • Algorithmic optimization and restructuring
Xilinx RTL tools	• FPGA architecture details • Basic RTL knowledge • RTL tools • Devices and interfaces
TI DSP tools	• Application expertise • C and assembly programming • Processor chip architecture details • Algorithmic optimization and restructuring • Devices and drivers

used with high-level synthesis tools. In addition, we found that the two high-level synthesis tools evaluated thus far (Synopsys Synphony C Compiler and AutoESL's AutoPilot) were able to achieve a level of resource-use efficiency comparable to that achieved using hand-written RTL code. While we did not directly evaluate the time savings afforded by using the HLSTs rather than writing RTL code by hand, we believe that the savings will be substantial, in part based upon our interviews with current HLST users.

FPGA designs created using traditional hand-written RTL coding typically take much more effort than the equivalent application implemented in software on a DSP processor. Therefore, perhaps the most surprising outcome of this project is that it took roughly the same effort to implement the evaluation workload on the FPGA (using either AutoPilot or Synphony C Compiler, plus the Xilinx tools) as it took on the DSP processor. This is a significant breakthrough, and one with the potential to have a major impact on the design of high-performance embedded computing applications.

Given FPGAs' advantages in speed and cost-performance in certain types of applications, we expect that the availability of competent high-level synthesis tools will significantly change the trade-offs between DSPs and FPGAs.

References

[1] Grant Martin, Gary Smith. "High-Level Synthesis: Past, Present, and Future," IEEE Design and Test of Computers, July/August 2009.