FPGA Illuminate: A Web-based Demonstration and Testing Application for Field Programmable Gate Arrays

Scott Edward Fields
University of Tennessee - Knoxville

Follow this and additional works at: https://trace.tennessee.edu/utk_chanhonoproj

Recommended Citation

This is brought to you for free and open access by the Supervised Undergraduate Student Research and Creative Work at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Chancellor's Honors Program Projects by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.
Appendix E - UNIVERSITY HONORS PROGRAM
SENIOR PROJECT - APPROVAL

Name: Scott Fields

College: Engineering Department: Electrical + Computer (ECE)

Faculty Mentor: Dr. Donald Bouldin

PROJECT TITLE: FPGA Illuminate: A Web-based Demonstration and Testing Application for Field Programmable Gate Arrays (FPGAs)

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: D. W. Bouldin Faculty Mentor

Date: May 2, 2004

General Assessment - please provide a short paragraph that highlights the most significant features of the project.

Comments (Optional):
Scott did an excellent job of selecting the appropriate means of implementing the desired functions. He made the user interface friendly, portable and secure. His report provides sufficient detail to allow another developer to add or improve on the present capabilities. He is to be commended for a job well done.

D. W. Bouldin
FPGA ILLUMINATE
A Web-based Demonstration and Testing Application for Field Programmable Gate Arrays (FPGAs)

Scott Fields
Advisor: Dr. Donald Bouldin

Senior Honors Thesis
5/3/2004

Microelectronics Laboratory
Dept. of Electrical and Computer Engineering
The University of Tennessee
Email: sfields1@utk.edu
ABSTRACT

BACKGROUND: Field Programmable Gate Arrays (FPGAs), being electrically programmable, offer many advantages over fixed chips. For one, they allow real-design testing and demonstration before delivery of a final design. However, software applications that provide these abilities are uncommon because chip designs vary greatly in functionality. Using several FPGA encryption cores as a starting point, a web-based application was developed with this aim. METHODS: Several core technologies were chosen to implement the application. HTML and JavaScript provide an interface that is accessible by any standard web browser. Perl and CGI run on a Linux/Apache web server to direct the user interaction. The server is secured by robust open-source software, password protection, and encryption. C/C++ drivers are called to interface to a Xilinx-based FPGA device. RESULTS: The framework was built upon to support several AES and SHA cores. Testing functions operate on inputs including strings, images, and files; additionally, the VHDL design specification can be inspected. The system is modular, allowing for future additions of features and new FPGA designs. Because of the technologies used, there is minimum dependence on the supporting hardware. CONCLUSION: FPGA Illuminate provides a web-based GUI for local or remote demonstration and testing of Field Programmable Gate Arrays. This ability fills a need and further increases the latent benefits of FPGAs over fixed designs.
Contents

1 Introduction .............................................................................................................. 4
2 Purpose ..................................................................................................................... 4
  2.1 Uses..................................................................................................................... 4
  2.2 Requirements ..................................................................................................... 4
    2.2.1 Accessibility ............................................................................................... 4
    2.2.2 Ease of use .................................................................................................. 5
    2.2.3 Extensibility ............................................................................................... 5
    2.2.4 Security ...................................................................................................... 5
    2.2.5 Portability .................................................................................................. 5
3 Design ....................................................................................................................... 5
  3.1 System Overview ................................................................................................. 5
  3.2 Server Hardware .................................................................................................. 7
  3.3 Server Software ................................................................................................... 8
    3.3.1 User Authentication ................................................................................... 8
    3.3.2 Secure Transmission .................................................................................. 8
    3.3.3 Accessibility ............................................................................................... 8
    3.3.4 Portability .................................................................................................. 8
  3.4 Presentation Layer ............................................................................................... 9
  3.5 Application Modules ........................................................................................... 9
    3.5.1 PERL ............................................................................................................ 9
    3.5.2 Loading FPGA Design ............................................................................... 9
    3.5.3 Error Checking ........................................................................................... 10
    3.5.4 File Uploads ............................................................................................... 10
  3.6 DriverS ................................................................................................................ 10
  3.7 FPGA Hardware ................................................................................................. 11
4 Results ..................................................................................................................... 11
  4.1 Login ................................................................................................................... 12
  4.2 Splash Screen ...................................................................................................... 12
  4.3 Application Window ............................................................................................ 13
  4.4 Load FPGA .......................................................................................................... 14
  4.5 AES ..................................................................................................................... 15
    4.5.1 Encrypt/Decrypt String ............................................................................... 15
    4.5.2 Encrypt/Decrypt Image ............................................................................... 16
    4.5.3 VHDL ............................................................................................................ 18
  4.6 SHA ..................................................................................................................... 18
    4.6.1 Hash String ................................................................................................. 18
    4.6.2 Hash File ..................................................................................................... 19
    4.6.3 VHDL ............................................................................................................ 20
5 Conclusion .............................................................................................................. 20
6 References .............................................................................................................. 22
Table of Figures

Figure 1: User interacts with the server over the internet via a secure connection ..............6
Figure 2: Chart shows the flow of control within FPGA Illuminate ......................................7
Figure 3: The Pilchard board plugs into a PC memory slot ......................................................11
Figure 4: A password challenge precludes access to FPGA Illuminate .................................12
Figure 5: A splash screen welcomes the user and resets the FPGA ....................................13
Figure 6: The main application window holds three panes: heading, table of contents, and application module interface ...........................................................................14
Figure 7: The countdown timer dynamically displays the time remaining until the FPGA is programmed ........................................................................................................14
Figure 8: The AES Encrypt/Decrypt String application module is shown after generating some results ..............................................................................................................16
Figure 9: The AES Encrypt/Decrypt Image application module is shown after decrypting an image using the FPGA ..................................................................................17
Figure 10: The VHDL design files for the FPGA can be browsed from the GUI .....................18
Figure 11: The SHA 256 Hash String application module is shown after a PC and FPGA result were generated .............................................................................................19
Figure 12: The SHA 256 Hash File application module is shown after generating the signature for a 1MB file ..............................................................................................20
Figure 13: The SHA 256 design files can be browsed from the GUI ........................................20
1 INTRODUCTION

Field Programmable Gate Arrays (FPGAs), being electrically programmable, offer many advantages over fixed chips. For one, they allow in-circuit testing and demonstration before delivery of a final design. However, software applications that provide these abilities are uncommon because chip designs vary greatly in functionality. Using a number of technologies, a web-based application was developed with this aim. In this paper, the aims and requirements will be outlined, the technologies used will be discussed, and the finished application, FPGA Illuminate, will be shown. FPGA Illuminate provides a web-based GUI for local or remote demonstration and testing of Field Programmable Gate Arrays.

2 PURPOSE

While FPGAs are capable of in-circuit testing and demonstration before a final design is produced, software applications that can function for the entire design space of FPGAs are rare. The aim of this author was to produce one such application that, while initially developed for encryption FPGA applications, would be applicable for most future designs.

2.1 USES

The main use of the software will be to demonstrate the FPGA designs both during remote and on-site presentations. Access to the demo may also be given to managers so that they can easily keep up-to-date on the progress of the project. These functions are important for communicating the progress of a very technical subject to both other engineers and to those with less-technical backgrounds.

Additionally, the application will be used for in-house testing. Testing is often done using command line C programs that pass data to the FPGA and check the results. A GUI provides a simple interface to the test bench, making rigorous testing less difficult. Also, since new technologies will be used in the design, it will be possible to leverage the various strengths of different technologies to ease future maintenance and development.

2.2 REQUIREMENTS

To provide the desired functionality, a number of requirements were specified for the design: accessibility, ease of use, extensibility, security, and portability. These requirements are elaborated below.

2.2.1 Accessibility

The application needed to be remotely accessible via the internet. Also, in the vein of the internet, the application needed to be accessible from a number of different platforms. A Java application, web-based application, or X-windows based application would meet these requirements. Furthermore, there needed to be a central server on which the physical FPGA could be located, and thus the method of connection had to make use of a client-server model.
2.2.2 Ease of use
For the purpose of non-supervised demonstrations, the interface needed to be self-explanatory, as access to the application may be granted along with no tutorial or other form of instruction. No new skills could be required to use the application. For supervised demonstrations, the interface also needed to be simple so as not to be distracting to the audience. Also, for the purpose of testing, the application needed to be an improvement over the command line, perhaps presenting options in an uncluttered graphical layout.

2.2.3 Extensibility
Ease of use for the developer was another concern for the project. FPGA development was an ongoing process, and so the source code needed to be easy to modify and update. The whole software development cycle, in fact, should be simple – easy to write, run, and debug. Not only was development expected to span many months and FPGA designs but also many different developers.

2.2.4 Security
As the internet is inherently insecure, the application needed to be secure at all levels. Basic access to the site needs to be restricted to authorized users only. Also, in-transit data could be sensitive and needed to be secured using strong encryption. Information assurance needed to be maintained while not hindering function.

2.2.5 Portability
Finally, with the likelihood of future system upgrades, the application needed to be portable. That is, at some time in the future the host computer would likely undergo changes to improve the hardware and/or software environment. Thus, the design needed to be both minimally dependent on the specific configuration and easily adaptable to accommodate any new system. Ideally, no code changes would be needed if either the operating system or the hardware platform were changed.

3 DESIGN
A web-based application was developed with the aim of meeting the requirements specified above. In the following subsections, the basic system is detailed through the top-down hierarchy of dataflow, and the technologies used in each level of the hierarchy are explored.

3.1 SYSTEM OVERVIEW
A user can access the application via any standard web browser, including but not limited to Microsoft Internet Explorer, Netscape Navigator, Mozilla, and Opera. The browser window then becomes the application window, allowing the user to give inputs to the program and view outputs. All communication between the user and the server is over secure channels, as shown in the figure below.
Dataflow within FPGA Illuminate follows the hierarchy shown below. Requests from the internet are dealt with by the server. The server program makes use of a CGI module plug-in that in turn runs and communicates with the PERL application modules. Application modules make use of drivers and algorithms coded in C, C++, and PERL to obtain their results. Finally, the drivers pass data to and from the FPGA, where the results of the input operations are calculated.
3.2 SERVER HARDWARE

An off-the-shelf Linux-supported PC was used as the server platform for this project. A desktop PC with an Intel PIII 933MHz processor and 256MB of RAM was chosen. The computer is connected to the University of Tennessee Ethernet network with a 100BaseT 100MBps Ethernet card. More powerful hardware is not necessary since the load on the machine is expected to be, and in fact limited to, a single web-based user.

A PC platform was chosen because it was supported by a desirable FPGA device, the Pilchard reconfigurable computing platform. On top of that, the choice to use off-the-shelf hardware for the server makes sense from a maintenance and support standpoint as well. The
popularity of PCs makes finding replacement parts and configuration tutorials relatively painless, leading to a cheaper system, a more stable system, and a system that allows its maintainers to focus on other tasks.

### 3.3 SERVER SOFTWARE

As will be discussed in the FPGA hardware section, the FPGA hardware device was designed to be used with the Linux operating system. Therefore, the Mandrake Linux distribution was chosen as the server operating system – specifically, version 8.2 of Mandrake with the 2.4.18-6mdk kernel was used [Man04]. The popular Apache HTTP web server, version 1.3, was then installed on top of that, and several modules were added to increase the usefulness of the server program [Apa04]. The Mandrake Linux and Apache server combination offered a distinct maintenance and configuration advantage due to their widespread use.

#### 3.3.1 User Authentication

With the default Apache HTTP web server installation, password authentication is supported. It allows any directory path on the server to be associated with .htaccess and .htpasswd files that specify usernames and passwords. Using this facility, the FPGA Illuminate web application was built to challenge users to verify their identity before granting them access to protected resources, providing the user authentication requirement listed above.

#### 3.3.2 Secure Transmission

The Apache server was enhanced via the mod_ssl module, which gives Apache the ability to use strong encryption [Eng04]. Information sent to and from the server is then protected by 128-bit Transport Layer Security (TLS) v1.0, a successor to the more familiar Secure Sockets Layer (SSL) scheme. With mod_ssl installed and configured, the encrypted HTTPS protocol is used to connect to the FPGA Illuminate website, rather than the unencrypted and insecure HTTP that is standard for most web sites. Thus, secure transmissions are ensured.

#### 3.3.3 Accessibility

Another module that was installed for Apache is mod_cgi, which allows the server to run Common Gateway Interface (CGI) scripts [Apa204]. CGI specifies a way in which inputs to a program can be passed through a URL and outputs can be displayed as a web page. Thus, a client can connect to the server with a web browser, can specify the name of the script and the script’s inputs in the URL line, and can view the output that the program generates. Since links and hidden fields can appear on a generated web page, a program can alter its state dynamically by changing what options are available based on the previous inputs. CGI is one of several popular methods for developing interactive web-based applications. It provides nearly limitless design options while not requiring extra user programs such as a Java installation or an X-windows server. Since all interactivity is available through a standard web browser, accessibility is guaranteed.

#### 3.3.4 Portability

The choice to use Apache is intelligent from the point of view of portability, because it can be installed on Linux, Windows, and Solaris operating systems. If in the future it is necessary to migrate to a different operating system, the server reconfiguration will be negligible. Additionally, with such a large user base, Apache promises to be fully-documented well into
the future, and it is relatively easy to find tutorials for resolving a number of common
difficulties.

3.4 PRESENTATION LAYER

There are a number of different languages and technologies that can be used to generate
displayable web pages, but FPGA Illuminate limits itself to HTML and JavaScript. These
two options are very mature and supported by nearly every internet browser, so questionable
accessibility is not an issue.

The basic formatting and data of the presented pages is HTML, while JavaScript provides
some special interactive functions. The basic intro page and frameset consist of static HTML
pages. The FPGA design listing is a JavaScript-generated menu that is created from a
configuration file each time the site is accessed. From the listing, different application
modules with different sets of options are selectable, and following any of these links passes
inputs to a CGI script that then generates pages to fulfill some function.

From an ease-of-use standpoint, HTML leaves little to be decided by the designer. HTML,
being designed for interfaces, has a standard look and feel that is familiar to any end user
who has ever used the internet. The addition of JavaScript provides extra feedback to the user
to explain what the application is doing. For instance, in addition to the standard browser
feedback of the loading icon and progress bar, JavaScript can provide extra information such
as a pop-up window that counts down through the delay associated with downloading a new
FPGA design.

3.5 APPLICATION MODULES

The application modules are the various CGI scripts that are run by the server. Application
modules are generally defined by their function and take the FPGA core design as an
argument. That is, there is a single application module that handles a task, such as encrypting
test strings or producing encrypted output images, and arguments such as the strength of the
encryption are passed to that module.

3.5.1 PERL

For several reasons, PERL was the chosen language for the application module CGI scripts.
First, PERL is in general the most common CGI language, so it is easy to find examples and
tutorials for completing a number of common tasks [Abo04]. Second, since PERL is a
scripted language, there is no compilation delay while working on new module designs.
Third, PERL is a very powerful language with many readily available modules for a number
of tasks, such as image processing and encryption [Cpa04]. Finally, PERL is historically used
in the chip design industry to tie together the multitude of design tools, so most FPGA
designers already have at least some rudimentary knowledge of it. All of these factors
combine to make PERL a favorable choice that promises to be easy to maintain while
growing with the requirements of the application.

3.5.2 Loading FPGA Design

One crucial function of every application module is the loading of the FPGA design into the
reconfigurable hardware. Since downloading a design takes approximately 14 seconds, it is
desirable that the application keeps track of the currently loaded design and only downloads a new design when it is necessary.

A state file is used to keep track of what design was last loaded to the FPGA, and when a user first enters FPGA Illuminate this file is cleared. When a user selects an action that requires a particular design, the state file is inspected to see if the required design is the loaded design or not. If the design is already loaded, then the action is performed. However, if the design needs to be loaded, then the FPGA is programmed and the state file is updated to reflect the loaded design file before the action is performed. Throughout one user session, many different FPGA designs can be programmed and tested.

3.5.3 Error Checking

Since users have the ability to enter strings, file paths, and other information, error checking user input is imperative to ensure a stable application. When inputs first enter the application, they are processed by a formatting function that checks for valid lengths and characters and ensures that the rest of the application can continue processing the inputs safely. If the formatting function makes minor changes, such as applying a default value or a truncating some input, then a warning is generated and outputted when the result is returned. If a more serious error is encountered, such as non-hex characters in a hex-only input field, then processing is aborted and an informative error message is returned to the user.

3.5.4 File Uploads

One essential feature of several application modules is their ability to operate on files. CGI provides this ability via a file upload dialog. First, the user clicks a file button, browses to a file on the local system and uploads it to the server. Next, the module saves the file as a temporary file so that it can operate on it, and the functions are performed. Finally, both the input and result can be displayed graphically through the interface. An example use of the file upload feature is that modules can be written to allow encrypting of images and to show both the input image and the garbled resulting image. Another example use is generating a secure hash signature for any input file rather than being limited to a short input string.

3.6 DRIVERS

The term “driver” is used broadly to refer to any program that returns operation results to the application modules. If the requested operation is to be run on the PC, then, the driver is generally a program coded in C, C++, or PERL. In the case of PERL, the driver may be embedded within the application module. For C and C++ programs, though, the drivers are standalone command-line-driven programs. Input to the driver can be passed over the command line or via a file that is referenced on the command line. Similarly, output from the driver can be returned via standard out (the terminal) or written to a file. In either case, the application module receives the output and can deal with it accordingly.

Low-level communication with the FPGA is also facilitated by C and C++ coded drivers that, like the PC-operation drivers mentioned above, are standalone command-line-driven programs. Generally each application module makes use of its own FPGA driver since the different cores and operations will have very different needs. FPGA drivers are similar in that they transfer 64-bit words to and from the Pilchard via the memory bus, instituting the proper
timing and handshaking to ensure correct outputs. As with the PC drivers, input can be via command line or file, and output can be via the terminal or through a file.

3.7 FPGA HARDWARE

The FPGA device attached to the server is the Pilchard, a board developed by the Chinese University of Hong Kong [Leo01]. A Xilinx Virtex1000E FPGA chip with approximately one million logic gates is the heart of the Pilchard. Pilchards are unique in that their interface is the standard PC 133MHz memory bus – the printed circuit board (PCB) of a Pilchard mimics the form factor of a memory DIMM, and a Pilchard is installed in one of the free memory banks of a PC (see figure).

Figure 3: The Pilchard board plugs into a PC memory slot

A programming cable is run from the serial port to the Pilchard, and this allows the FPGA design to be loaded. As part of any design, the hardware interface to the memory bus can be implemented, and once this is done then communicating with the Pilchard simply becomes a matter of reading and writing data to the appropriate memory addresses. Since the memory bus is 128bits wide, communication is high-bandwidth, and performance can be quite high.

Pilchards were designed for use with the Linux operating system, and that is how they were implemented in this project. A Pilchard device is installed as a logical device in the Linux file system, and then data can be read from and written to it as if it were any other file. Downloading FPGA designs to the Pilchard is possible with the ‘download’ command. Designs to be downloaded can be generated using any industry-standard synthesis tool, such as Synplicity, Leonardo Spectrum, or Synopsys.

4 RESULTS

The aforementioned design has been implemented for two families of security designs: the Advanced Encryption Standard (AES) and the Secure Hash Algorithm (SHA). The following sections explain how the user interacts with FPGA Illuminate and how the application’s capabilities have been leveraged to provide an interface for the AES and SHA cores.
4.1 LOGIN
Upon trying to access the TLS-secured web page of FPGA Illuminate, the user is initially challenged to provide a valid username and password (see figure below). The query takes the form of a standard browser password request, and upon entering the correct credentials, access is granted to the site. The username and password are encrypted for transmit so unauthorized individuals will not be able to “sniff” the password.

![Password Challenge]

**Figure 4:** A password challenge precludes access to FPGA Illuminate

4.2 SPLASH SCREEN
After supplying correct login credentials, the user initially sees a splash screen showing the Pilchard reconfigurable computing platform and the University of Tennessee logo (see figure). As is evident by the icon and text bubble in the upper left, encryption is enabled, so the site is secure to use. The user may click on the image or the links beneath it in order to continue into the main application. This splash screen is also the page that, unbeknownst to the user, clears the state file of which FPGA was previously loaded. Clearing this memory at the beginning of each session ensures that other programs can freely reprogram the FPGA at other times without disrupting the function of FPGA Illuminate.
Figure 5: A splash screen welcomes the user and resets the FPGA

4.3 APPLICATION WINDOW

The main application window, shown below, consists of three panes. The top pane is simply a title bar, the left pane shows the table of contents where the FPGA core and function can be chosen, and the right pane shows the application module interface.
The table of contents is structured as a tree, and it makes managing a large number of FPGA designs simple by supporting a hierarchal organization. In the figure it can be seen that both the AES and SHA cores have been implemented and that there are a number of different AES cores and a single SHA core. For each of these cores, several functions are available for selection.

4.4 LOAD FPGA

As was discussed in the design section above, the user receives a visual indicator of the delay associated with reprogramming the FPGA hardware. This indicator is shown in the figure below and consists of a basic countdown timer that closes itself when the process completes, allowing the user to continue interacting with the main application that is now capable of utilizing the FPGA.

Figure 7: The countdown timer dynamically displays the time remaining until the FPGA is programmed.
4.5 AES

AES encryption performs a series of operations on an input block of 128 bits and returns the encrypted result. AES is becoming increasingly popular and has been approved by the National Institute of Standards and Technology (NIST) for use in information handling by the federal government. This method of encryption is marked by its variable strength, relative speed, and ability to be implemented in hardware [Nis04]. The FPGA cores supported by FPGA Illuminate make use of key lengths of 128, 192, and 256 bits.

4.5.1 Encrypt/Decrypt String

The first application module for AES cores is the Encrypt/Decrypt String module that provides the ability to encrypt or decrypt either a hex or ASCII text input string. The interface for the Full AES core is shown in the figure below after being exercised. Several sample inputs and outputs are listed at the bottom of the interface (only one is visible in the figure), and set of these was passed through both the PC and CORE drivers. A 128 bit key was used and the input was in hexadecimal. The operation was encryption, and the test mode was Monte Carlo, meaning that the operation was looped ten thousand times before returning the result. The outputs from the PC and CORE operations are shown below the input, and they are both verifiably the same as the sample result. Any other inputs, of course, could have been used and checked against one another.
Full AES Encrypt/Decrypt

Key Size:  
- 128  
- 192  
- 256  

Key (hex): 6a5964499d35cea7e1ac707b37b923ed

Input Type:  
- hex  
- text  

Input Block: (16 bytes) a3d43bfa65d0e80092f67a314857870

Operation:  
- encrypt  
- decrypt  

Test Type:  
- Monte Carlo (10,000x)  
- Known Answer  

Source:  
- PC  
- CORE  

Output Type:  
- hex  
- text  

Monte Carlo Codebook Samples:

128 bit Key: 6a5964499d35cea7e1ac707b37b923ed  
Plaintext: a3d43bfa65d0e80092f67a314857870  
Ciphertext: 355f697e8b868b65b25a04e18d782afa

Figure 8: The AES Encrypt/Decrypt String application module is shown after generating some results.

4.5.2 Encrypt/Decrypt Image

The AES Encrypt/Decrypt Image application module allows a more visual representation of the encryption and decryption operations by means of images. In the figure below, the input image is the result of a 256 bit encryption that was performed via the PC. The result of this was saved and then uploaded to the server for processing by the FPGA. As can be seen, the original image data was recovered, thus verifying the function of the FPGA and demonstrating the interoperability potential of the FPGA as part of an AES-based cryptography scheme.
<table>
<thead>
<tr>
<th>Key Size:</th>
<th>128</th>
<th>192</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key (hex):</td>
<td>3404909800abd3558439acd0834123433404909800abd3558439acd0f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Image:</td>
<td>pc_encrypted.png</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>encrypt</td>
<td>decrypt</td>
<td></td>
</tr>
<tr>
<td>Source:</td>
<td>PC</td>
<td>CORE</td>
<td></td>
</tr>
</tbody>
</table>

Note: A small 150x150 image takes ~30 seconds to process, and larger images take minutes. Please be patient and do not click Reload or Stop.

Sample Image 1: test1.png test1_encrypted.png (encrypted with 128-bit key: 11111111111111111111111111111111)
Sample Image 2: test2.png test2_encrypted.png (encrypted with 128-bit key: 1234567890123456789012345678900)

Figure 9: The AES Encrypt/Decrypt Image application module is shown after decrypting an image using the FPGA
4.5.3 VHDL

While the aforementioned AES application modules show the functionality of the FPGA, users may also be interested in the underlying logic design. To this end, the VHSIC Hardware Design Language (VHDL) files are available to be browsed through the FPGA Illuminate GUI. A listing of the files is presented, as shown below, and the full VHDL source is available for inspection.

- pilchard.vhd
- parith.vhd
- pcore.vhd
- RIJNDAEL Top Iterative.vhdl
- rijndael pkg.vhdl
- interface.vhd
- key sched iterative.vhd
- alg iterative.vhd
- controller_iter.vhd

Figure 10: The VHDL design files for the FPGA can be browsed from the GUI

4.6 SHA

SHA is another algorithm approved by NIST for use by government agencies. SHA, however, generates a message digest, or a compressed signature of some data [Nis204]. There are several different strengths of SHA, and the only FPGA design that is currently supported by FPGA Illuminate is SHA-256, which gives 256 bit output strings.

4.6.1 Hash String

Hash values for strings of any length can be generated by the SHA Hash String application module, shown in the figure below. The figure shows that one of the test strings with a provided hash result was passed as input to the application. The PC result was first generated, followed by the FPGA result. The 256 bit outputs were checked against each other and against the sample results and verified to be correct.
SHA 256 Hash String

SHA Strength: 256

Input: abc
cdecdefgghighijhijklm

Source: PC, CORE

Generate Result

PC Output: 24BD6A61 D20638B8 E5C02693 0C3E6039 A33CE459 64FF2167 F6ECE004 19DB06C1
FPGA Output: 24BD6A61 D20638B8 E5C02693 0C3E6039 A33CE459 64FF2167 F6ECE004 19DB06C1

Sample Hashes:

input: abc
hash: BA7816BF 8F01CFFA 414140DE 5D3E2223 B00361A3 96177A9C B410FF61 F20015AD
input: abc
cdecdefgghighijhijklm
hash: 24BD6A61 D20638B8 E5C02693 0C3E6039 A33CE459 64FF2167 F6ECE004 19DB06C1

Figure 11: The SHA 256 Hash String application module is shown after a PC and FPGA result were generated

4.6.2 Hash File

The SHA 256 Hash File application module enables the GUI to return the hash value for any arbitrary file. This function is useful because the user can then easily verify the result of the FPGA against any other known good implementation of the SHA 256 algorithm. In the figure shown below, a 1 MB file consisting of one million ‘a’ characters was used as input, and the output was tested both using the C algorithm on the PC and using the design loaded into the Pilchard device. Once again, the results are shown to be correct, demonstrating the truth of results returned by the FPGA design.
SHA 256 Hash File

SHA Strength: 256
Input File: million_a.txt
Sample File (one million 'a's): million_a
Source: CPC CORE

PC Output: CDC76E5C 9914FB92 81A1C7E2 84D73E67 F1809A48 A497200E 046D39CC C7112CD0
FPGA Output: CDC76E5C 9914FB92 81A1C7E2 84D73E67 F1809A48 A497200E 046D39CC C7112CD0

Sample Hashes:
input: ‘a’ x 1000000
hash: CDC76E5C 9914FB92 81A1C7E2 84D73E67 F1809A48 A497200E 046D39CC C7112CD0

Figure 12: The SHA 256 Hash File application module is shown after generating the signature for a 1MB file

4.6.3 VHDL

So that the user can explore the logical design of the FPGA, the VHDL can be browsed from the GUI. This feature has many advantages. For one, it can allow a technical user to delve into the functionality of the design. For a non-technical user, it may serve as reassurance that the design actually works, since the results given by the GUI could theoretically have been generated solely by software algorithms. Finally, the VHDL browser can aid in organizing the design project, creating a central repository for all of the completed project files.

Figure 13: The SHA 256 design files can be browsed from the GUI

5 CONCLUSION

This project has provided a useful framework to facilitate web-based verification and demonstration of FPGA designs. The application framework was built upon to support
several AES and SHA security cores, resulting in the ability to encrypt/decrypt/hash strings, encrypt/decrypt images, generate hash strings of files, and browse the VHDL files for all designs. The application provides the previously lacking function of a generic software application framework for remote testing and demonstrations of FPGAs. With the qualities of accessibility, ease of use, extensibility, security, and portability, FPGA Illuminate is an aesthetically pleasing and remarkably versatile tool for the designers of FPGAs.
6 REFERENCES


