Implementation and verification of SKIPJACK Algorithm using Verilog

Aamir Muhammad, Member, IEEE

Abstract—The results of design, implementation and verification of the secret-key block cipher SKIPJACK algorithm using the RTL level Verilog HDL and FPGA platforms. This project is a good task for getting a good insight into the working of secret key ciphers and to explore better ways of implementation with the help of different optimization techniques. Only the codebook mode of operation is used in the project to keep the effort to a reasonable level.

Index Terms—Cryptography, Algorithm, encoding, decoding, Ciphertext, codes.

1. INTRODUCTION

Hardware implementation of cryptography will thrive in the new century because of growing requirements for high speed, high volume secure communications combined with physical security. Very few results regarding hardware implementation of the SKIPJACK algorithm have been published so far.

In this article, the main focus on implementing and comparing SKIPJACK algorithm using Field Programmable Gate Arrays (FPGAs). This technology referred to as reconfigurable hardware, offers many advantages for the future vendors and users of cryptography equipment. It assures a short time to the market, high flexibility (including a capability for the frequent modifications of hardware) and low development cost. The same integrated circuit can be used for the execution of cryptographic algorithm.

Abbreviation and Acronyms
FPGA Field Programmable Gate Arrays
VerilogHDL Verilog Hardware Descriptive Language

2. ALGORITHM DESCRIPTION

The NSA developed the SKIPJACK algorithm in 1993 for a family of hardware-based crypto-processors used by a variety of government and military agencies for confidentiality services. It was kept secret until 1998, when the details of the approach were declassified. The algorithm is an iterative, symmetric-key block cipher with a block size of 64 bits, a key size of 80 bits, and a fixed 32 round structure. The parameters for the algorithm are presented below in Table 1. Each of these parameters is fixed to its predetermined value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Block Size in bits</td>
<td>64</td>
</tr>
<tr>
<td>K</td>
<td>Key Size in bits</td>
<td>80</td>
</tr>
<tr>
<td>R</td>
<td>Number of Rounds</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 1 SKIPJACK Parameters

Table 2. Modes Of Operations

<table>
<thead>
<tr>
<th>Modes Of Operation</th>
<th>Block Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Feed-Back (OFB) Modes</td>
<td>64</td>
</tr>
<tr>
<td>Cipher Feed-Back (CFB) Modes</td>
<td>64/32/16/8</td>
</tr>
<tr>
<td>Codebook</td>
<td>64</td>
</tr>
<tr>
<td>Cipher-Block Chaining (CBC)</td>
<td>64</td>
</tr>
</tbody>
</table>

Only the codebook mode as shown in the Figure 1 is implemented for this project.

Encryption

The SKIPJACK encryption algorithm is fundamentally composed of two stepping rules entitled Rule A and Rule B. Each rule operates on four 16-bit words (w1…w4) composed the block size of 4 × 16 = 64 bits. The words are shifted,
exclusive ORed, and G-transformed (See section 0) according to the permutations in the rule. These rules are repeated 8 times and alternated such that the encryption stepping sequence is 8 * Rule A, 8 * Rule B, 8 * Rule A and 8 * Rule B making a total of 4 × 8 = 32 rounds.

The counter value to be used in the permutations is initialized to 1 at the beginning of the encryption sequence. K is the round number.

**Rule A**

Fig. 2 depicts the permutation for Rule A of SKIPJACK encryption.

![Graphical representation of Rule A](image)

Fig. 2. Graphical representation of Rule A

The stepping Rule A can be written in equation forms as:

\[
\begin{align*}
  w_1^{k+1} &= G^k (w_1^k) \oplus w_4^k \oplus \text{counter}^k \\
  w_2^{k+1} &= G^k (w_1^k) \\
  w_3^{k+1} &= w_2^k \\
  w_4^{k+1} &= w_3^k
\end{align*}
\]

**Rule B**

Fig. 3 depicts the permutation for Rule B of SKIPJACK encryption.

![Graphical representation of Rule B](image)

Fig. 3. Graphical representation of Rule B

The stepping Rule B can be written in equation forms as:

\[
\begin{align*}
  w_1^{k+1} &= w_4^k \\
  w_2^{k+1} &= G^k (w_1^k) \\
  w_3^{k+1} &= (w_1^k) \oplus w_2^k \oplus \text{counter}^k \\
  w_4^{k+1} &= w_3^k
\end{align*}
\]

**Decryption**

SKIPJACK decryption algorithm is composed of two stepping rules. In this case, the rules are the inverse of the encryption rules entitled **Rule A Inverse** and **Rule B Inverse**. The inverse rules are also stepped 8 times each in an alternating fashion yielding 32 rounds composed of 8 * Rule B Inverse, 8 * Rule A Inverse, 8 * Rule B Inverse, and 8 * Rule A Inverse.

![Graphical representation of Rule A inverse](image)

The counter value is initialized to 32 at the beginning of the decryption sequence. K represents the round number.

**Rule A^{-1}**

Fig. 4 depicts the permutation for Rule A\(^{-1}\) of SKIPJACK decryption.

![Graphical representation of Rule A inverse](image)

The stepping Rule A\(^{-1}\) can be written in equation form as:

\[
\begin{align*}
  w_1^{k-1} &= [G^{-1}]^{k} (w_2^k) \\
  w_2^{k-1} &= w_3^k \\
  w_3^{k-1} &= w_4^k \\
  w_4^{k-1} &= w_4^k \oplus w_2^k \oplus \text{counter}^{k-1}
\end{align*}
\]

**Rule B^{-1}**

Fig. 5 Graphical representation of Rule B\(^{-1}\)

The stepping Rule B\(^{-1}\) can be written in equation form as:

\[
\begin{align*}
  w_1^{k-1} &= [G^{-1}]^{k} (w_2^k) \\
  w_2^{k-1} &= [G^{-1}]^{k} (w_2^k) \oplus w_3^k \oplus \text{counter}^{k-1} \\
  w_3^{k-1} &= w_4^k \\
  w_4^{k-1} &= w_1^k
\end{align*}
\]

**G-Permutation**

Each stepping rule contains a G-transform for each of the 8 rounds. The G-transform, seen as the heart of the algorithm, is a cryptovariable (input key) dependent 4-round Feistel structure. The F function is a simple permutation lookup table. In the following description, each g value (g1 … g6) is 1 byte. The input bytes g1, and g2 are the high and low bytes of the 16-bit input word respectively.

The permutation is recursive and composed of the mathematical relationships given below.

\[G^k (w = g_1 || g_2) = g_5 || g_6\]
Where,
\[ g_i = F (g_{i-1} \oplus cv_{4k+i}) \oplus g_{i-2} \]

\( k = \) step number (the first step is 0)
\( F = \) Substitution table (See section 0)
\( cv_{4k+i} = \) (4k+i)\textsuperscript{th} byte in the cryptovariable schedule

Therefore:
\[ g_3 = F (g_2 \oplus cv_{4k}) \oplus g_1 \]
\[ g_4 = F (g_3 \oplus cv_{4k+1}) \oplus g_2 \]
\[ g_5 = F (g_4 \oplus cv_{4k+2}) \oplus g_3 \]
\[ g_6 = F (g_5 \oplus cv_{4k+3}) \oplus g_4 \]

Similarly, for the inverse \( [G^k]^{-1} \)
\[ [G^k]^{-1} (w= g_5 \mid g_6) = g_1 \mid g_2 \]

where
\[ g_2 = F (g_{i-1} \oplus cv_{4k+i}) \oplus g_i \]

The schematic representation of the G-transform for both encryption and decryption is given below in Fig. 6. In the diagram, CV stands for crypto-variable that is taken as the key. The subscript of CV decides the byte location in the 10-byte key for the particular transform.

Fig. 6. G-Permutation Diagram

### Key Scheduling

The SKIPJACK algorithm uses an 80-bit key in its natural order. Thus, no permutation of the key is performed. The keys used in the internal Feistel structure of the G-transform are accessed on a per byte basis using a simple direct array-addressing scheme modulo the number of bytes (i.e. 10 bytes). This simple operation is done inline with the algorithm and thus, no separate key scheduling is needed.

### F-Table

The SKIPJACK permutation look-up table (F-table) is given in Fig. 7. The high order 4 bits of the input index the row and the low order 4 bits index the column. For example, \( F(7A) = D6 \).

![F-Table](image)

### 3. SKIPJACK VERILOG RTL DESIGN

#### Overview

The RTL level implementation of the SKIPJACK algorithm is designed as an I/O device with 16-bit general-purpose asynchronous processor interface. Input and output buffers, key and control signals are handled through the memory mapped registers in the module.

One of the key features of the implementation is the unified module for both encryption and decryption operation. The mode of operation is controlled by using encryption decryption bit in control register.

Data buffers and processing of the algorithm is designed in such a way that the new data can be written to the input buffer while the previous data is being processed. This scheme maximizes the throughput of the device in overall system.

Another aspect of the design is the use of the technology specific look-up ROMs that are used to implement the F-Permutation tables as well as the look up for cryptovariable index for Feistel structure. These ROM LUTs (Look Up Tables) are generated by the Altera and Xilinx design software. For synthesis through the Synplify program, these LUTs are marked as synthesis black box.

#### TOP Level Block Diagram

As shown in Fig. 8 there are two main modules in the SKIPJACK implementation.

1. Register Memory Module
2. Algorithm Module

The figure also shows the external interface signals of the SKIPJACK implementation along with the bus interface control logic that is part of the top-level SKIPJACK module.
Register Memory Module
Register memory module implements the mechanism to interface the internal registers with the external signals. Regmem module has memory mapped registers for input and output buffers, key buffer and the control flags.

Algorithm Module
This module is the core for implementing the SKIPJACK algorithm. A single module is used for both the encryption and decryption operation. Under command from the external device, it takes the data from the input buffer, performs encryption or decryption depending upon the encryption/decryption flag and stores the result in the output buffer.

![Top-level block diagram](image)

**Fig. 8 Top-level block diagram**

Interface
SKIPJACK module interfaces with the external device using memory mapped registers via 16-bit bus interface. Input/output from the device as well as the operation is controlled through the read and writes to the internal registers.

As shown in Error! Reference source not found., there are 12 word (16-bit) registers for data interface and one 16-bit register for control of the device.

**Device Input**

**Input Buffer:** Four-word (64 bits) register is used to load data into the module for encryption or decryption.

**Crypto-Variable:** Five-word (80 bits) register is used to load the private key into the module.

**Mode flag:** Bit [4] of the control register is used as a flag bit to control the operating mode of the design

1 for encryption and
0 for decryption.

**Start Flag:** Bit [0] of the control register is used to start the encryption or decryption operation

**Device Output**

**Output Buffer:** Four-word (64 bits) register is used to read the encrypted or decrypted data from the module

**Done Flag:** Bit [1] of the control register is used as a done bit to indicate completion of the last operation

Verilog Modules for SKIPJACK
For clarity and flexibility, six Verilog modules are used for Implementing the SKIPJACK algorithm in FPGA.

Table. 3 Verilog Modules
Detailed Description

Following subsections provide a detailed description of the SKIPJACK Verilog RTL design.

SKIPJACK Module

SKIPJACK module is the top-level module for implementing the SKIPJACK algorithm in RTL level Verilog. Module inputs and outputs are the I/Os of the SKIPJACK device and are mapped to the actual pins of the FPGA.

Two main sub-modules regmem and algorithm are instantiated in this module and the signals are wired between the two sub-modules.

This module has logic for the external bus interface. A simple 6 level state machine is used to control the READY signal to the external processor and to generate the internal single cycle write pulse to the regmem module for storing the data. The bi-directional data bus is internally split into input and output data bus. This module implements the buffer with control logic to interface the input and output buses with the external bi-directional bus.

Regmem module

Regmem module has code to implement 12 registers. Write to the individual registers is controlled by the write pulse from the SKIPJACK module and the address on the bus interface. All the registers have a read-back capability. Data for the read operation is selected by a 16-bit 1:12 asynchronous multiplexer that uses the address inputs to select the content of one of the internal register.

Algorithm Module

The algorithm module is the core for the SKIPJACK encryption and decryption. As shown in Error! Reference source not found., main logic functions of the algorithm module are:

1. Processing Registers P-regx
2. Counter
3. P_regx compute
4. Control state machine
5. Feistel (G and G⁻¹ function)

All the logic functions with the exception of the Feistel are coded within the algorithm module. Feistel function is instantiated as a sub-module with the algorithm module.

All the functions in the module and sub-module are designed to execute the encryption or decryption function based upon the encrypt/decrypt flag in the control register.

### Table: Module Details

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skipjack.</td>
<td>SKIPJACK module is the top-level module of the Verilog RTL implementation. This module has bus interface logic and links the two main modules of the design; regmem and algorithm.</td>
</tr>
<tr>
<td>regmem.</td>
<td>Register memory (regmem) module has implementation for data input, data output, key input and control register.</td>
</tr>
<tr>
<td>algorithm</td>
<td>Algorithm module has implementation for overall encryption/decryption engine including: Control state machine, K-counter and data holding registers. It uses the Feistel module as a sub-module.</td>
</tr>
</tbody>
</table>

### Figure 10: Block Diagram of the Algorithm Module

**Processing Registers P-regx**

The processing registers are used during the encryption or decryption steps (32 cycles of Rule A and B). Input from the input registers W-IN1 through W-IN4 is loaded into the P-regx before computations. The result of the encryption or
decryption is latched into the output registers W-OUT1 through W-OUT4 after the completion of the operation.

This mechanism provides the flexibility to read the previously computed data and write the new data for processing while another set of data is being processed. This in effect creates a pipeline mechanism for data interface, thus, increasing the throughput of the overall system.

**Counter**

The algorithm module uses a loadable up/down counter to count the number of cycles for the encryption or decryption rules. For encryption mode, the counter is initialized with 0 and counts up to 31 before rounding back to 0. For decryption mode, the counter is initialized with 31 and counts down to 0 before rounding back to 31. Counter increments or decrements under control from the state machine.

Following section depicts the Verilog code for the counter.

```verilog
always @(posedge clk or negedge resetn)
begin
  if (~resetn)
    k_cnt <= 5'h00;
  else
    begin
      if (cntr_init && enc_decn)
        k_cnt <= 5'h00;
      else if (cntr_init && !enc_decn)
        k_cnt <= 5'h1F;
      else if (inc_cntr)
        k_cnt <= k_cnt + 1;
      else if (dec_cntr)
        k_cnt <= k_cnt - 1;
    end
end
```

The bit #4 of the counter is used to decide between the Rule A and Rule B. This is used by the compute engines for selecting the next output along with the encryption or decryption mode.

The output of the counter is added with 1 before using it in the compute function, as the algorithm requires the value to be between 1 and 32.

**P_regx compute**

P_regx compute is used to load the W-INx value during the INIT phase of the encrypt/decrypt operation. It is also used to load the next value of the W-INx register after each iteration. The next value to be loaded is based upon the encrypt/decrypt flag and the rule selection A or B.

For example, the Rule A of encryption for W1 is:

\[ w_{1}^{k+1} = G^{k+1} (w_{1}^{k}) \oplus w_{4}^{k} \oplus \text{counter}^{k} \]

Rule B of encryption for W1 is:

\[ w_{1}^{k+1} = w_{1}^{k} \]

Rule A\(^{-1}\) of decryption for W1 is:

\[ w_{1}^{k-1} = [G^{k-1}]^{-1} (w_{2}^{k}) \]

Rule B\(^{-1}\) of decryption for W1 is:

Combining all the operations, the Verilog code for the P_reg1 compute is:

```verilog
always @((posedge clk or negedge resetn))
begin
  if (~resetn)
    p_reg1 <= 16'h0000;
  else
    begin
      if (alg_state_machine == `INIT)
        p_reg1 <= w_in1;
      else if (inc_cntr || dec_cntr)
        begin
          case ({enc_decn, mode_B_An})
            2'b10 : p_reg1 <= f_out ^ p_reg4 ^ k_cnt_val;
            2'b11 : p_reg1 <= p_reg4;
            2'b00 : p_reg1 <= f_out;
            2'b01 : p_reg1 <= f_out;
          endcase
        end
    end
end
```

**Control State Machine**

A 5-state state machine is used to control both the encryption/decryption process. This state machine is used as trigger to load the P_regx register, initialize K-counter, control the K-counter operation (number of iterations [32] for encrypt/decrypt function), control the done bit, and load the output register with the final P_regx value.

State machine and in turn the encryption/decryption operation is triggered by writing a 1 to the start bit in the control register.
Feistel Module

Feistel module implements the code to compute G and G⁻¹ function based upon the enc_decn control bit. As shown in Error! Reference source not found., Feistel function is a completely combinatorial function implemented using ROM Look Up Tables (LUTs) and math function. ROM LUTs are specific to the FPGA technology and are generated separately for the Altera and Xilinx FPGA and instantiated as sub-modules within the Feistel module. Clut and the Flut are the only technology specific modules in the implementation.

The Feistel structure uses four F-permutation tables that are implemented by four 256 x 8 ROM LUTs. Each level of the Feistel structure uses one byte (8-bit) of the 10-byte (80-bit) cryptovariable. The index is based upon the k-counter with an offset, 0, 1, 2, 3 and is computed modulo 10. To simplify the complexity and avoid modulo 10 addition, the 4 index values for each level are pre-computed for each value of K and are stored in a 32 x 16 ROM LUT. This can also be accomplished by using CASE statement in Verilog, as there are only 5 values for the 16-bit index output.

As the function is implemented using only the combinatorial logic, the level of logic limits the speed of the clock that can be used. The clock speed can be increased by putting registers at each level. However, as the input to each level is dependent upon the previous level, the structure cannot be unrolled and parallelized.

3. FUNCTIONAL VERIFICATION

DESCRIPTION

The functional verification of the module is done using the Silos III simulation software. A top-level Verilog testbench (skipjack_tb) is implemented to verify the operation of the Verilog implementation. Test bench generates the read and writes to the top-level module (skipjack).

Overall operation was verified by using the plaintext, cryptovariable and Ciphertext test vectors provided by the NIST. Plaintext input: 33221100ddcbbaa Cryptovariable: 00998877665544332211 Ciphertext out: 2587cae27a12d300

Silos III software provides visibility into not only the top-level signals but also into all the signals used within the implementation. This has been very useful when debugging the mistakes in the sub-module design.

Each step of the operation was verified and the sub-modules were debugged using the intermediate vector data for each step provided in the vector table from NIST. See Appendix A for vector.

Simulation Captures

Encryption

<table>
<thead>
<tr>
<th>Name</th>
<th>Scope</th>
<th>Value</th>
<th>0.003us</th>
<th>0.223us</th>
<th>0.446us</th>
<th>0.669us</th>
<th>0.892us</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reset</td>
<td>.k_fb</td>
<td>STB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>.k_fb</td>
<td>STB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>state[5:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl1_sel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl2_sel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl3_sel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvl4_sel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 11 Control State Machine

Feistel Module

Fig. 12 Block Diagram of the Feistel Module

The module implements both the G and G⁻¹ function with unified code. Feistel module operates on W1 for encryption function and on W2 for decryption function.

G⁻¹ can be computed using the G Feistel structure but flipping bytes at the input and output, feeding the LSB (instead of the MSB) of the 16-bit value to the right arm of the Feistel structure. Also, the CV byte used at each level is flipped with encryption using the k-cntr + 0 value for level 1 and decryption using the k-cntr + 3 value. This is done by selecting the appropriate 4-bit index values from the LUT for each level based upon the encrypt/decrypt mode. Thus for Feistel structure level 1, the multiplexer selects the index value k-cntr + 0 for encryption and k-cntr + 3 for decryption.
1. **PLAINTEXT WRITE AND ENCRYPT COMMAND**

**Description**

Verilog RTL implementation of the SKIPJACK algorithm is verified by synthesizing the code and loading it on an actual FPGA. As a first step the code is targeted to an Altera 1K50-3 FPGA on an evaluation module. Static timing analysis tool is used to calculate to maximum possible operating speed of the implementation.

**Synthesis and PLACE&ROUTE**

Altera MaxPlus II, version 10.1 is used for both synthesis and place&route function. Inputs and outputs from the top-level skipjack.v module are mapped to the pins of the FPGA with 16-bit data bus and asynchronous interface to the processor on the evaluation module.

**SYNTHESIS RESULTS**

Chip/Device: EP1K50FC256-3  
Input Pins: 8  
Output Pins: 1  
Bidir Pins: 16  
Memory Bits: 8704 (1088 Bytes) [256 x 8 x 4 + 32 x 16]  
LCs used: 1038  [each LC is 4-bit LUT with 1 FlipFlop]  
(Logic Cell)

%Utilized: 36%

Therefore, estimated gate count for the implementation is **18K FPGA Gates**.

**Throughput**

Using the static timing analyzer of the MaxPlus II tool the maximum operating frequency reported is **10.41MHz**.

The comparatively lower speed is the result of the combinatorial implementation of the Feistel structure. Total delays through the Feistel computational stage are:

- 4 ROM LUT Delays + 6 XOR delays + 2 mux-select delays.

Maximum speed of the operation can be increased by using registers between each step of the Feistel structure. However, such an implementation will require 4 clock cycles per iteration.

Based upon the maximum FPGA speed and assuming that the data is read and written to the FPGA while the previous operation is in progress is:

\[
\text{Clock speed} / (\text{number of clocks}) \times \text{number of bits} = 10 \times 6/34 \times 64 = 20Mbps
\]

**Verification**

Actual verification of the SKIPJACK implementation is done by using an evaluation module with 1K50FPGA operating at 10MHz. This evaluation module has serial interface and provides simple memory modification commands to control the read and writes to the FPGA.

The SKIPJACK device bit file evaluation module is downloaded to the FPGA through the JTAG header by the MaxPlus II software. Then the Windows hyper terminal communication software is used to communicate with the FPGA board.

Registers inside the SKIPJACK device are read and written through the memory modification commands. Test vectors and the key are written and the operations are then controlled in this way through the PC.

**CONCLUSION**

The results and analysis presented in this paper shows the design, implementation and performance of SKIPJACK algorithm in FPGAs.

**APPENDIX A: TEST VECTORS**

All Values are hexadecimal. All information is presented with the most significant bit/Byte/words to the left.

**SKIPJACK-CODEBOOK MODE**

Plaintext: 33221100ddccbbaa
Cryptovariable: 0099887766554432211
Intermediate steps:
Ciphertext Output: 2587cae27a12d30

ACKNOWLEDGEMENT

The author would like to thanks Prof. Dr. Kris Gaj, Prof. Dr. Tarek El-Ghazawi and their students in helping me in this project.

REFERENCES


AUTHOR

Muhammad Aamir; Dept of Electrical and Computer Engineering; The George Washington University; Washington, DC20052 USA.