FPGA Implementation and Analysis of an Error Correction Code for Physical Uncloneable Functions

Brian Jarvis
ECE 646
12/09/2014
Application and Motivation

What is a Physical Unclonable Function (PUF)?
- IC-specific identification
- Unique response by exploiting manufacturing variations
- Ability to provide stimuli to the PUF to produce device-specific responses

PUFs can provide
- IC identity via challenge and response schemes
- Secure key generation

Why use PUFs?
- Difficult to predict what key will be, given physical access
- Extremely difficult/unlikely to duplicate PUF ID generation without exact hardware
What’s the problem?

PUF ID generation reliability
Not all bits reproduced identically every time
For example:
  • Iteration 1: 1010101010101010
  • Iteration 2: 1010101110101000
Can be sensitive to temperature, voltage, and other environmental factors
Error correcting codes are needed!
Convert noisy values into reliable ID
Implementations typically expected to provide bit-error rate of $10^{-6}$ or less
Selected implementation must use minimal area
BCH Code

Cyclic linear block code

Pros

• Capable of very low probability of error
• Capable of correcting many errors in large messages
  • For example, 18 errors corrected in a message size of 131 bits (124 parity bits).
• Simple encoder design as LFSR

Cons

• Complex decoder
• Large code words have large slice utilization
amro BCH Thesis


C application with parameter configuration file

Uses templates and C BCH algorithm to generate VHDL

Pros
• Very simple to use
• Generates simulation testbench

Cons
• Reliant on C application
• Output files use some poor coding styles
• Output files are not well organized
  • 3000 lines of VHDL for decoder
amro VHDL coding improvements

Signal types: BIT replaced with STD_LOGIC
  • Also STD_LOGIC_VECTOR now used instead of BIT_VECTOR

Clock process events: (clk’event and clk=‘1’) replaced with rising_edge(clk)

Edits done in C application and in template files used in VHDL code generation

Process sensitivity lists added

Components not reliant on C algorithms pulled out into stand-alone VHDL files
amro BCH Timing analysis

Encoder:

- N clock cycles for complete code word generation
  - K clock cycles to run message through LFSR
  - N-K clock cycles to drain LFSR

Decoder:

- Parallel Architecture
  - N + 56 clock cycles (311 for N=255) to first output
  - Additional K clock cycles to drain error correction buffers and correct errors

- Serial Architecture
  - N + 182 clock cycles (437 for N=255)
  - Additional K clock cycles to drain error correction buffers and correct errors

K = Message size
N = Code word size
N – K = Number of parity bits
Example implementation parameters:
- Message input size = 131
- Code word length = 255
- Correctable errors = 18

Parallel Architecture on Kintex 7 XC7K70T-FBG676:
- 882 slice registers
- 2694 slice LUTs
- 1350 occupied slices

Serial Architecture on Kintex 7 XC7K70T-FBG676:
- 1007 slice registers
- 997 slice LUTs
- 464 occupied slices

Parallel uses more complex structure in exchange for wait states.

BCH Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>N=511, k=241, t=36</th>
<th>N=255, k=131, t=18</th>
<th>N=127, k=64, t=10</th>
<th>N=15, k=5, t=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>1933</td>
<td>882</td>
<td>447</td>
<td>114</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>5429</td>
<td>2694</td>
<td>1112</td>
<td>264</td>
</tr>
<tr>
<td>Occupied Slices</td>
<td>2496 (24%)</td>
<td>1350 (13%)</td>
<td>577 (5%)</td>
<td>(1%)</td>
</tr>
</tbody>
</table>
MicroBlaze implementation

MicroBlaze platform and application created with Xilinx Platform Studio and SDK

BCH C application
- Instruction size: 11310 bytes
- Data size: 3568 bytes

Profile with 16 KB instruction and 16 KB data
- Kintex 7 XC7K70T-FBG676
  - 1532 slice registers
  - 2131 slice LUTs
  - 855 occupied slices (8%)
Repetition Codes

Simplest class of error correction code

Each message bit repeated to desired code length

For example, if repetition factor is 3:
- Message: 1100101   Code word: 11111100000111000111

Encoding and decoding is very simple in hardware
- Encoder can be realized with just wires
- Decoding is a simple majority function

Capable of correcting floor(N/2) errors
Concatenated Codes: Rep[13,1] • BCH[15,5]

Powerful error correcting capability from product of two codes
- Minimum distance of each is multiplied for exponential decrease in error probability

Error probability meet or exceeds that of stand-alone large-area BCH
- Bit-error probability less than $10^{-7}$ for above example

Area is greatly reduced by using much smaller BCH code
- $(15, 5, 3)$ BCH implemented on Kintex 7 XC7K70T-FBG676
  - 141 Slice registers
  - 204 Slice LUTs
  - 81 Occupied Slices
Concatenate Code Caveats

Repetition-based concatenated codes carry additional entropy-loss
Koeberl et al. derive more conservative entropy figures

<table>
<thead>
<tr>
<th>Method</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$l_2$</th>
<th>PUF size</th>
<th>Entropy</th>
<th>$P_{\text{Fail}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard FE</td>
<td>$R[3, 1, 3]$</td>
<td>$BCH[977, 232, 205]$</td>
<td>3</td>
<td>8793</td>
<td>256</td>
<td>1.9 E-7</td>
</tr>
<tr>
<td>hard FE</td>
<td>$R[5, 1, 5]$</td>
<td>$BCH[474, 204, 73]$</td>
<td>3</td>
<td>7110</td>
<td>256</td>
<td>3.1 E-8</td>
</tr>
<tr>
<td>hard FE</td>
<td>$R[7, 1, 7]$</td>
<td>$BCH[817, 542, 57]$</td>
<td>1</td>
<td>5719</td>
<td>256</td>
<td>4.9 E-7</td>
</tr>
<tr>
<td>hard FE</td>
<td>$R[7, 1, 7]$</td>
<td>$BCH[460, 289, 41]$</td>
<td>2</td>
<td>6440</td>
<td>256</td>
<td>7.1 E-7</td>
</tr>
<tr>
<td>soft FE[23]</td>
<td>$R[8, 1, 8]$</td>
<td>$G[24, 12, 8]$</td>
<td>107</td>
<td>20544</td>
<td>256</td>
<td>3.4 E-6</td>
</tr>
</tbody>
</table>

For at least 128 bits of leftover entropy
- Much larger BCH codes: Between 460-977 bit N
- Alternative inner code: Reed-Muller or Golay code

Conclusions

BCH Codes have very good error correction performance
BCH Decoder is very complex
  • Area scales linearly with error correcting capability
  • 24% of Kintex 7 XC7K70T-FBG676 occupied for N=511, k=241, t=36
BCH implementation in software on MicroBlaze becomes more attractive solution beyond 12 errors corrected
Concatenating BCH with Repetition allows both area-efficiency and error correcting performance
Further reduction in area may be realized with alternate inner code
  • Reed Muller or Golay Code
  • Cryptographic integrity maintained with the above inner codes to avoid requiring very large BCH codes