Error Correcting Code Encoder and Decoder for Physical Unclonable Functions

Analysis, Software Modeling, and FPGA Implementation

Brian Jarvis

ECE 645

Spring 2014

Application and Motivation

- What is a Physical Unclonable Function (PUF)?
  - IC-specific identification
  - Unique response by exploiting manufacturing variations
- PUFs can provide
  - IC trustworthiness
  - Secure key generation
  - Protection against physical attacks

What's the problem?

- PUF signature generation reliability
- Not all bits reproduced identically every time
- For example:
  - Iteration 1: 1010101010101010
  - Iteration 2: 1010101110101010
- Error correcting codes are needed!
- Convert noisy bit- or value- stream into reliable signature
- Selected implementation must yield an error rate less than what the PUF started with
- Selected implementation must use minimal area

Possible Approaches Analyzed

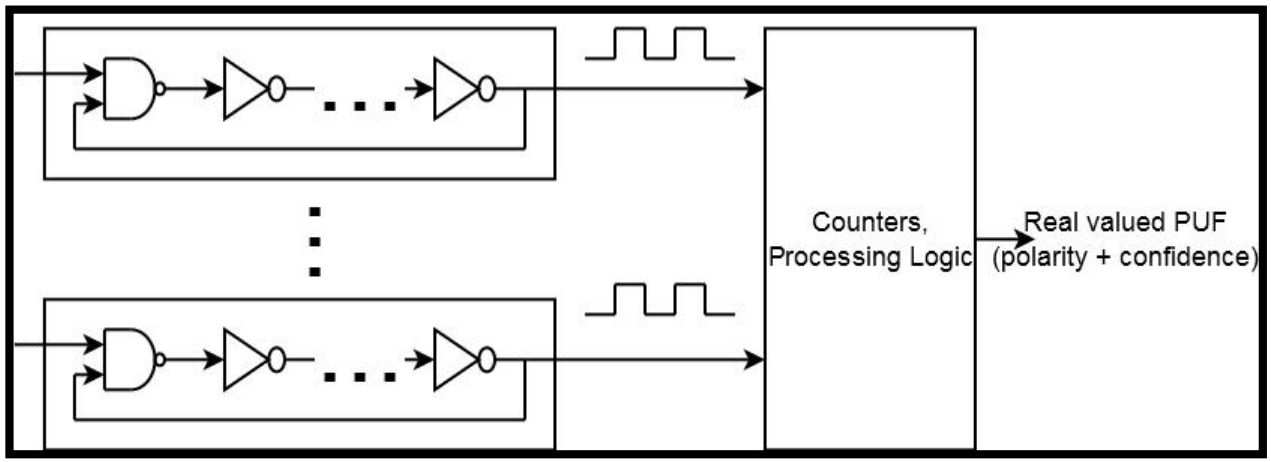
- Maes et al. compares performance of combinations of coding techniques.
  - Optimal combination: Reed Muller Code + 4x Repetition.
  - Maes, R.; Tuyls, P.; Verbauwhede, I., "A soft decision helper data algorithm for SRAM PUFs," *IEEE International Symposium on Information Theory, 2009. ISIT 2009.*, pp.2101-2105, June 28 2009-July 3 2009
- Maes et al. describe the use of BCH codes in support of secure key generating PUF circuit.
  - R. Maes, A. Van Herrewege, I. Verbauwhede, "PUFKY: A Fully Functional PUF-based Cryptographic Key Generator," Proc. CHES 2012, LNCS 7428, pp. 302-319

Previous Work

- Yu and Devadas propose Index Based Syndrome (IBS) encoding and decoding.
- State that BCH codes which provide reasonable correction are too large (area) to be practical
- IBS encoder and decoder described in 2010 paper:
  - Meng-Day Yu; Devadas, S, "Secure and Robust Error Correction for Physical Unclonable Functions," Design & Test of Computers, IEEE , vol.27, no.1, pp.48-65, Jan. -Feb. 2010
- Publication includes results from implementation on Xilinx Virtex 5
- Implementation for this project will focus on smaller Spartan 3 and Spartan 6 FPGAs

Background: Ring Oscillator PUF (ROPUF)

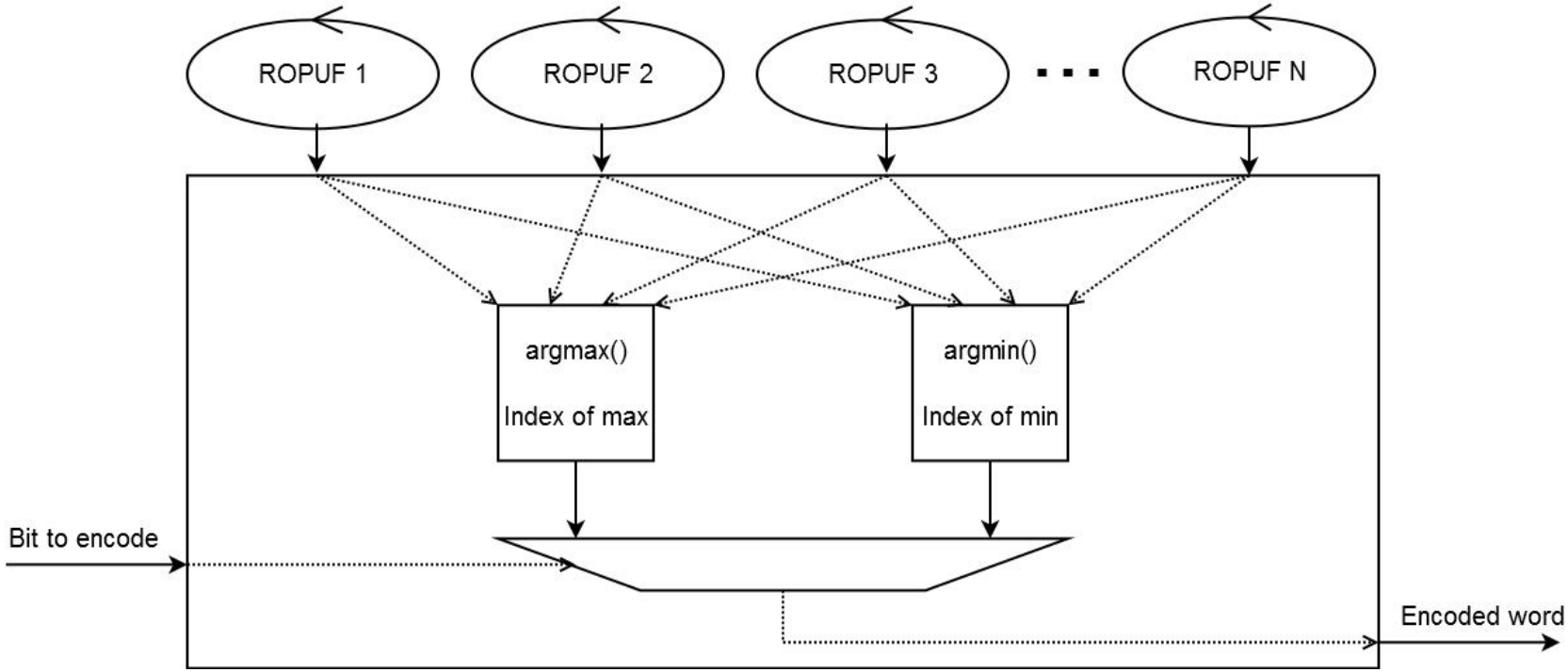
- Digital logic loops which are designed to oscillate at a particular frequency (can be configurable)
- Manufacturing variations yield oscillation frequencies which differ independently on each chip
- Combined with counters, accumulation, and processing logic can produce a real valued PUF output



Source: Suh, G.E.; Devadas, S, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," 44th ACM/IEEE Design Automation Conference, 2007. DAC '07., pp.9-14, 4-8 June 2007

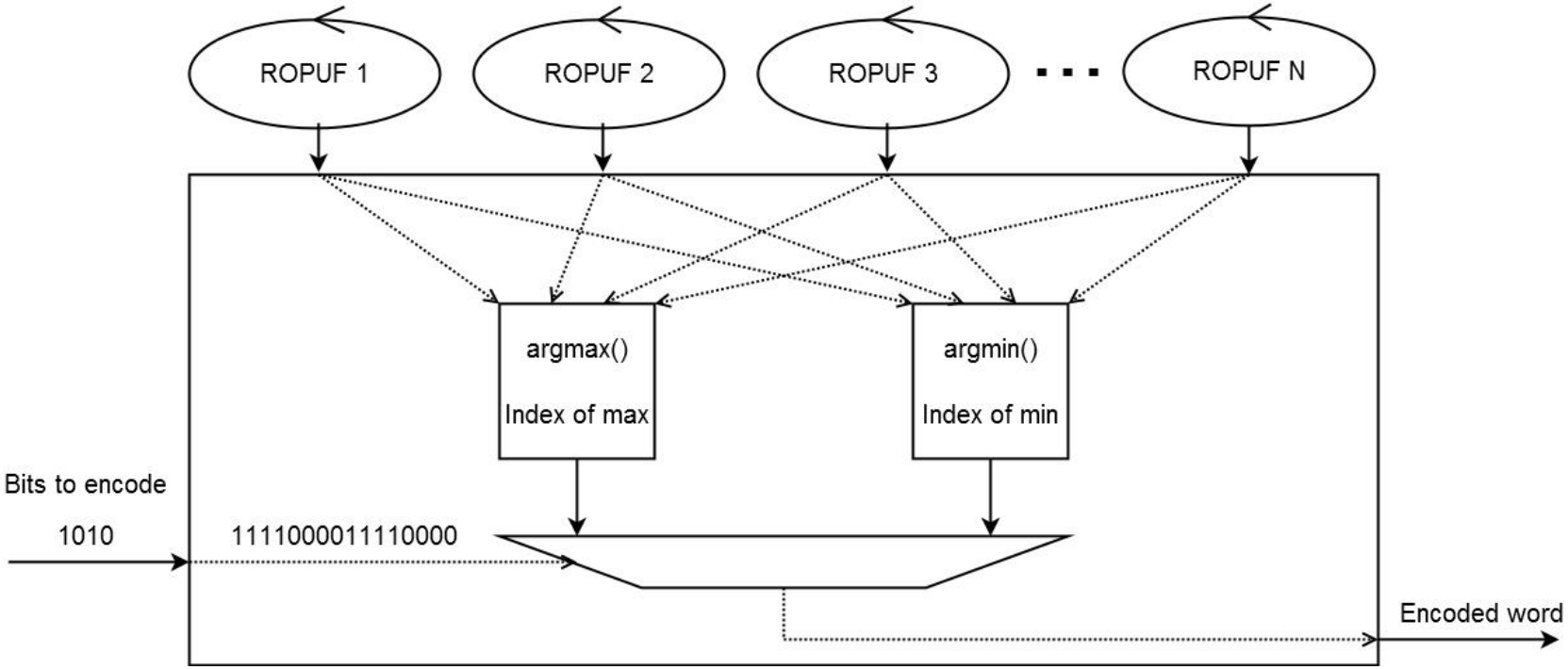
Algorithm: IBS Enmapping

- N ROPUF generators provide RV-PUF results for comparison
- Bit to encode dictates whether maximum or minimum is selected
- Index of max or min (width $\log_2(N)$) becomes encoded word



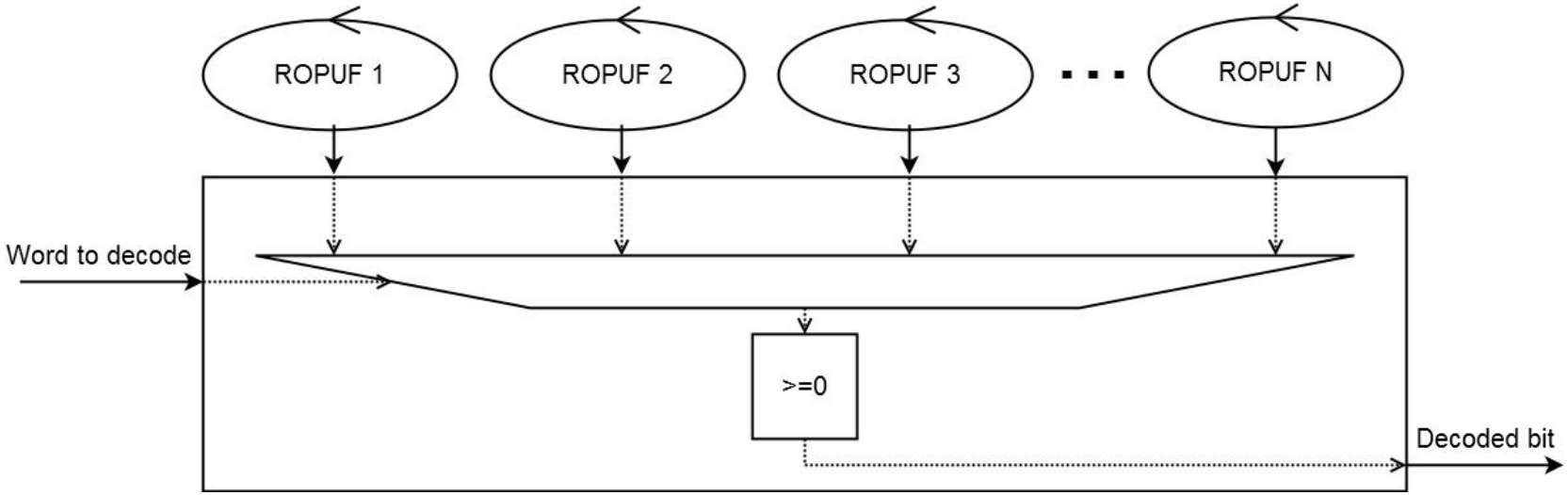
Algorithm: IBS Enmapping with Repetition Coding

- Repetition coding can be applied to further improve theoretical error correction performance
- Illustrated with 4x repetition coding



Algorithm: IBS Demapping

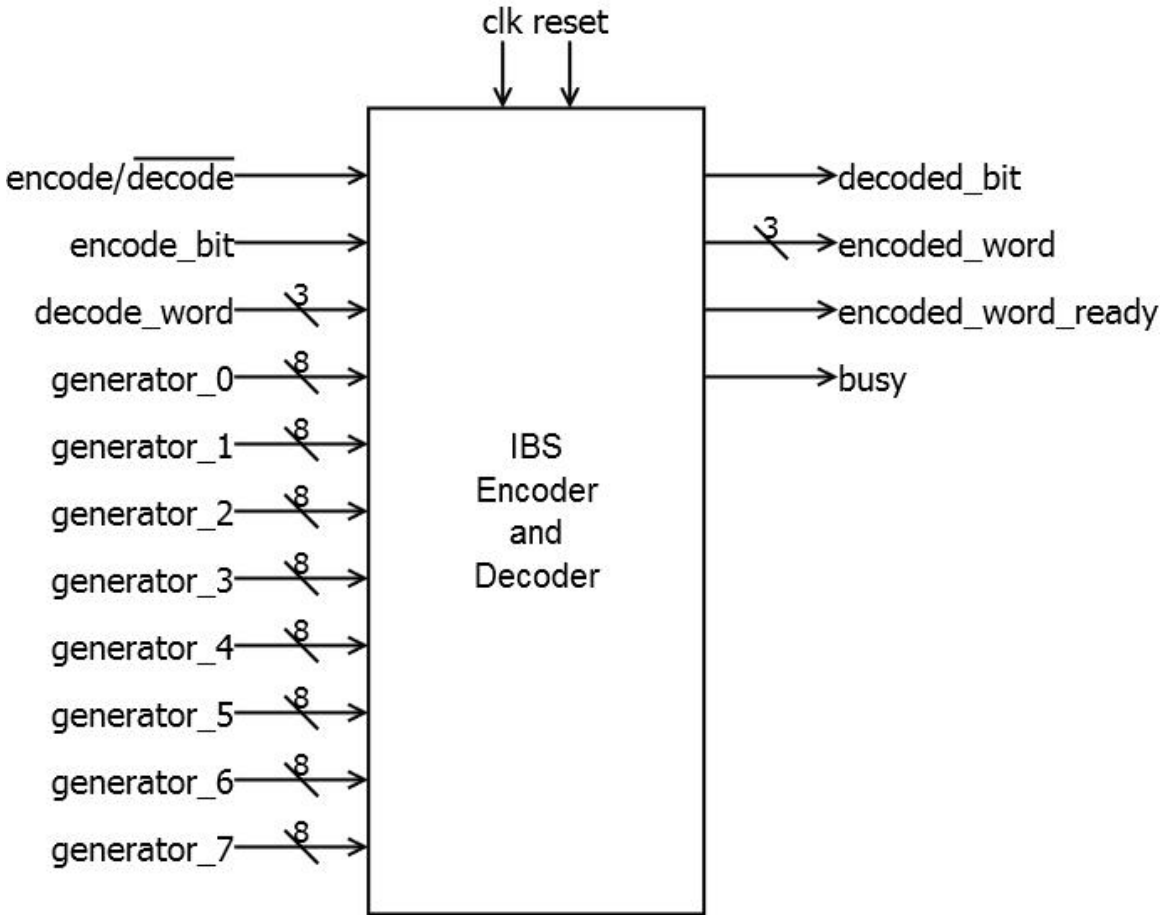
- N ROPUF generators provide RV-PUF results for comparison
- Word to decode selects one of the N RV-PUF results
- Compare RV-PUF results for negativeness
  - >= 0 decodes a 1 bit
  - < 0 decodes a 0 bit



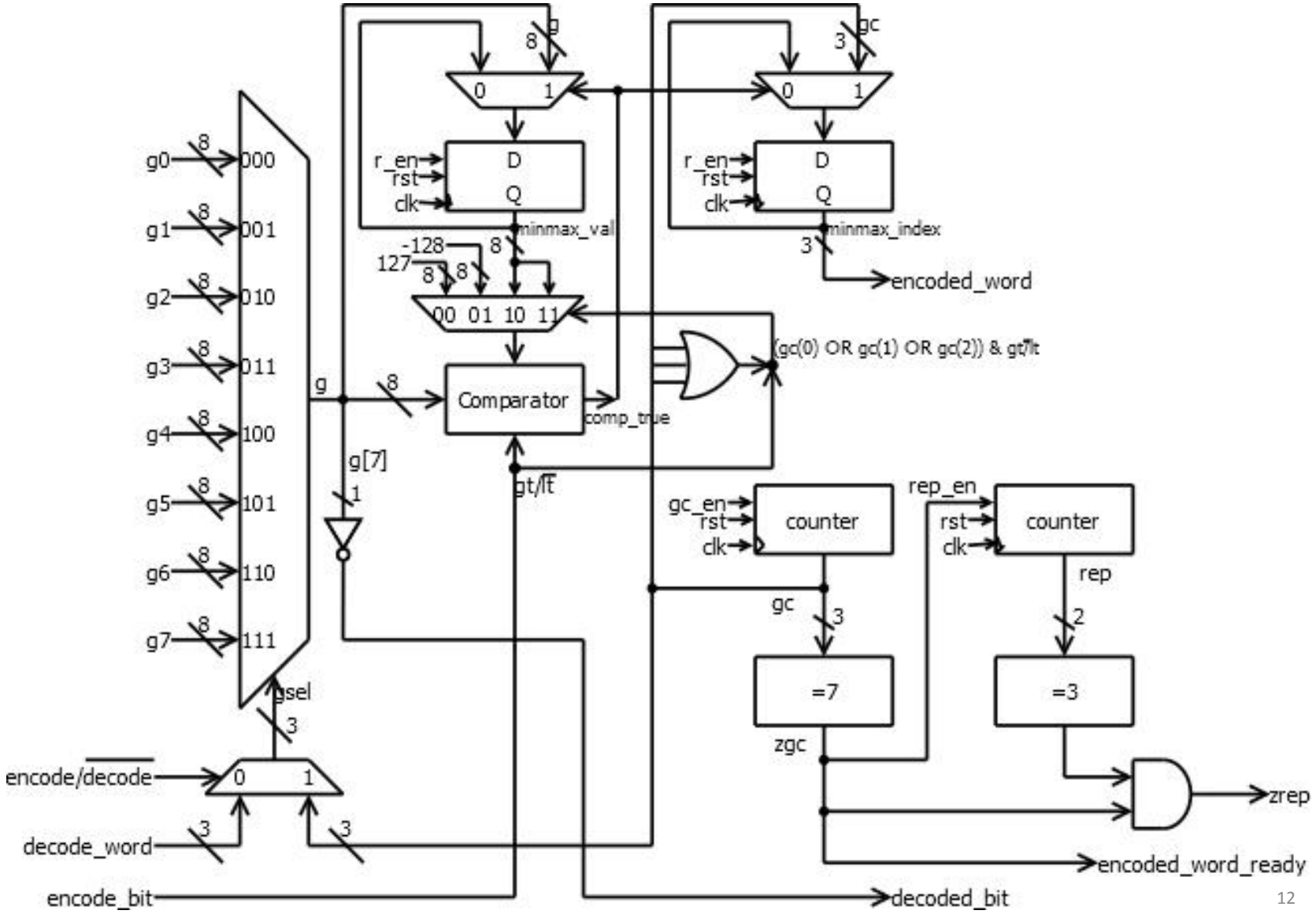
Software Modeling

- Python implementation developed
- RV-PUF Generator “base” values simulated using pseudorandom number generator
- Base values “fudged” each iteration to simulate successive RV-PUF results varying slightly
- 100% success rate after five 1000-iteration simulations:
  - Eight 8-bit generators, 1024 encode bits
- 0% success rate when generator “base” values forced to be in range [-20,20].
- Shows that algorithm is effective when generators have significant dynamic range

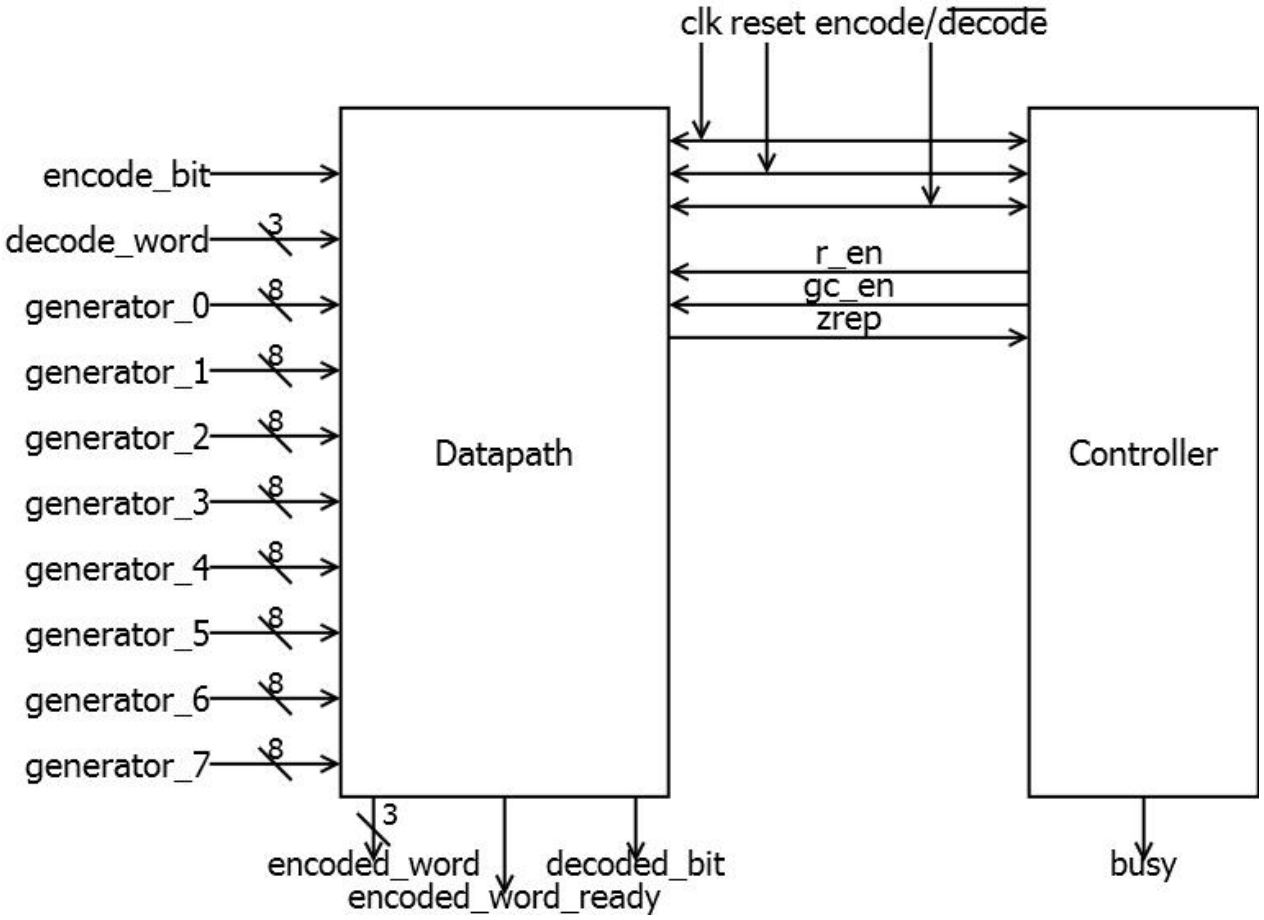
Top Level Interface



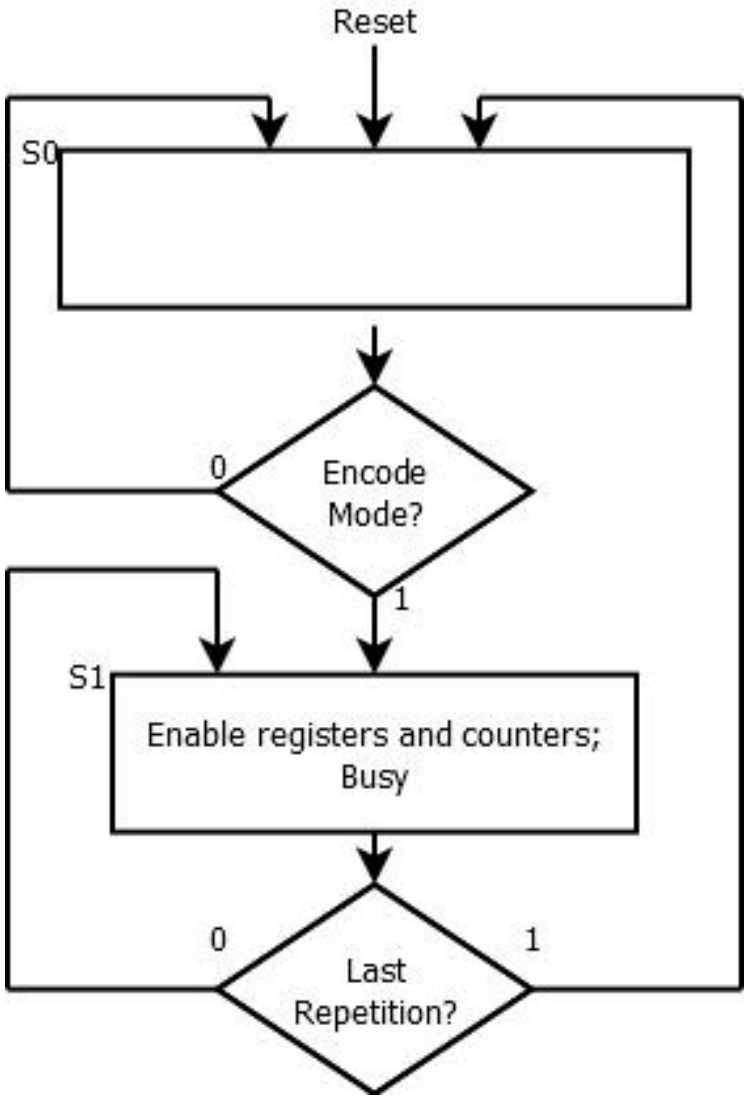
Block Diagram



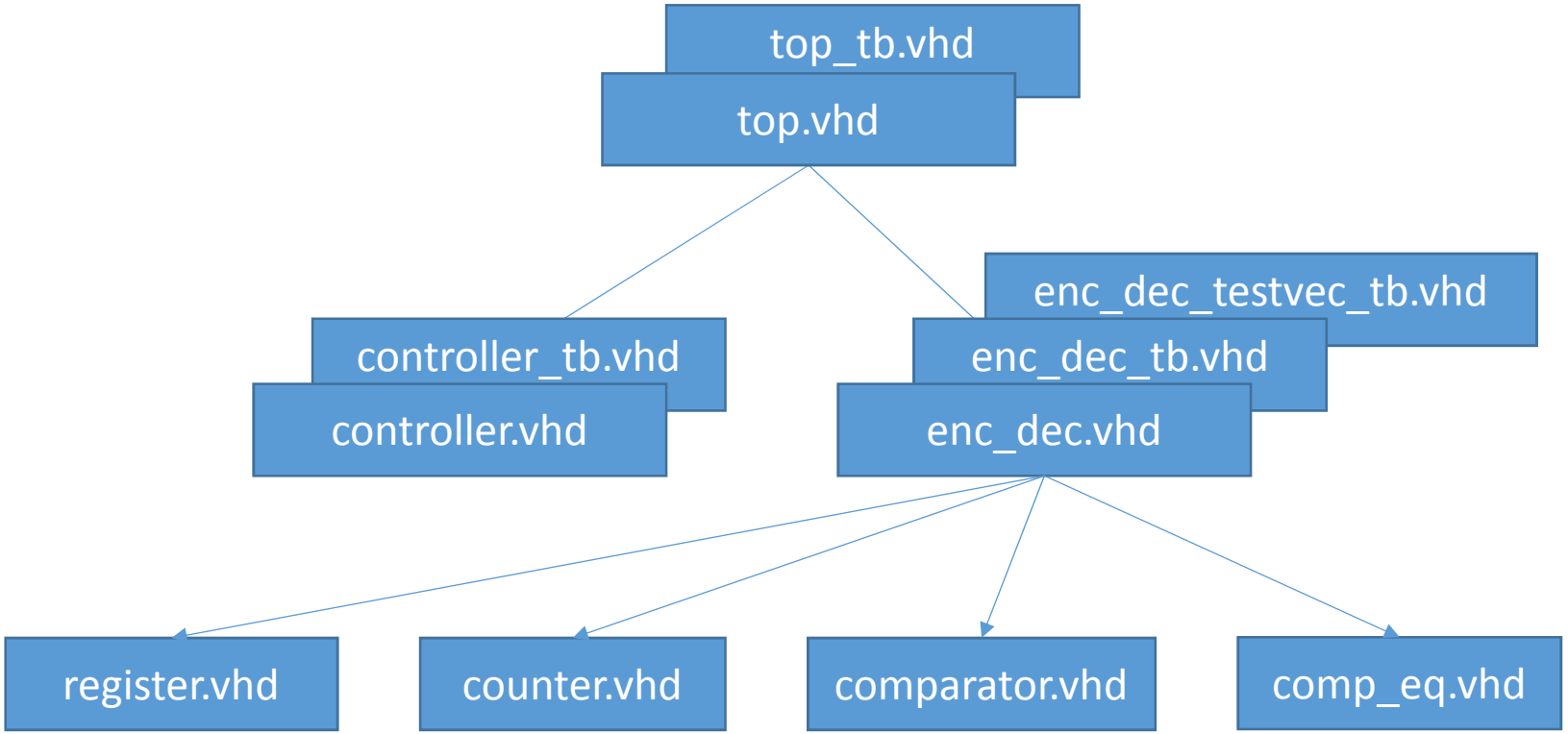
Interface: Datapath and Controller



ASM Chart


Reset
S0
0
Encode
Mode?
1
S1
Enable registers and counters;
Busy
0
1
Last
Repetition?

Source Code Chart



All source code verified in functional simulation

Test Vectors

- Python simulation also used for generating test vectors
- Three files generated:
  - File with generator values per iteration
    - 8-bit values in hex format, space-delimited
  - File with bits to encode
    - 1-bit values, space-delimited
  - File with expected encoded words
    - 8-bit values in hex format, space-delimited

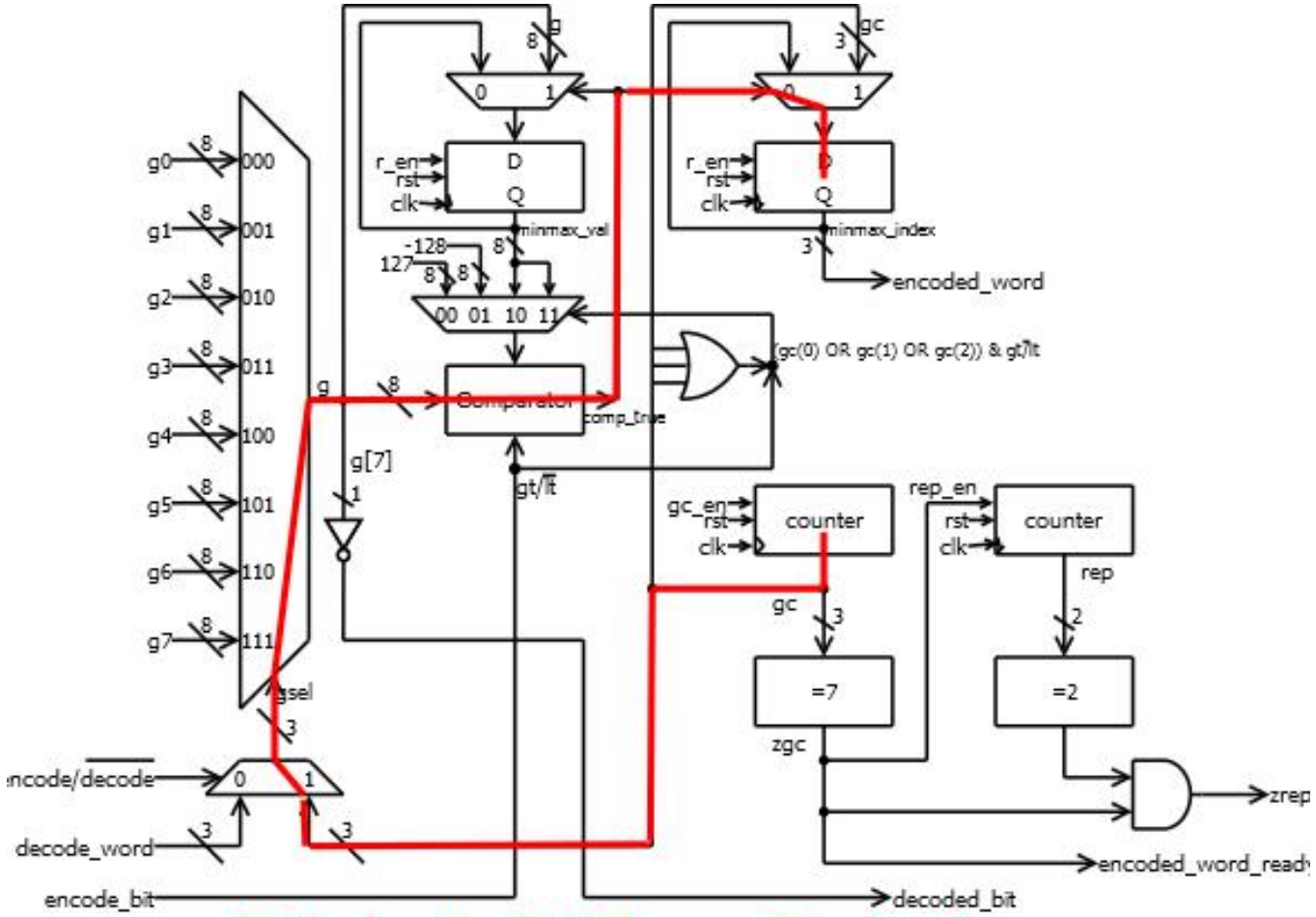
Test Benches

- Higher complexity components tested with testbenches
  - Datapath, controller, and top level
- Two forms of testbench used for datapath
  - Testvector file-based testbench exercises inputs and verifies against expected outputs using a process loop
  - Simple stimuli-based testbench used during preliminary testing as well as for testing the decode path due to its simplicity
- Stimuli-based testbench used to verify controller operation
- Stimuli-based testbench used to verify top-level operation

Timing Analysis

- $Latency_e$
  - By design, the latency for encoding is determined by number of ROPUF inputs and repetition factor: 8 * 4 in this implementation = 32 clock cycles
- $Latency_d$
  - The decoding circuit is fully combinational logic, thus the latency is 1 clock cycle if registers surround the circuit
- Successive input timing
  - Because each input bit requires the latency time in order to encode, successive encoding inputs require 32 clock cycles (# ROPUFs * repetition factor)
  - Decoding can be performed either at combination logic speed, or at the speed of one decode per clock cycle if registers surround the circuit
- $Throughput_e = 1/(32*T_{clk})$ encodes per second
- $Throughput_d = 1/T_{clk}$ decodes per second

Timing Analysis: Critical Path



Critical path: 8.017 ns on Spartan 6
9.333 ns on Spartan 3

Results: Spartan 3 XC3S50 and Spartan 6 XC6SLX4

	Spartan 3		**Spartan 6**	
Resource Utilization (Slices)	41		42	
Resource Utilization (percent)	5.34%		1.75%	
Maximum Clock Frequency (MHz)	107.15		124.73	
Minimum Clock Period (ns)	9.33		8.02	
	Encode	**Decode**	**Encode**	**Decode**
Minimum Latency (ns)	298.66	9.33	256.54	8.02
Maximum Throughput (bits/second)	3,348,333.87	107,146,683.81	3,897,966.82	124,734,938.26
Throughput / Area (bps/slice)	81,666.68	2,613,333.75	92,808.73	2,969,879.48

Conclusions and Lessons Learned

- Tradeoff between Area and Speed
  - Faster approach was also first instinct: tree of comparators to evaluate all PUF values simultaneously
  - Design chosen which favors smaller logic area
  - Throughput suffers from sequential evaluation for min/max, but this fits the concept of operation
- Caveats
  - Implementation is designed for PUF values which contain polarity and confidence values
  - Large IO resource utilization due to parallel load of PUF values
    - Sequential load design possible, but requires external logic for decode
- Index based syndrome is a good choice for PUF encoding and decoding
  - Very small in terms of FPGA slices needed to implement
  - Ability to augment with repetition coding provides additional error tolerance
  - Generics allow for customizable repetition coding and PUF value bit widths