April 2013 Reference Design RD1179

RC4 Based PRNG Generator

Introduction

RC4 is a pseudo random sequence generation algorithm developed in 1987 by Ron Rivest for RSA Data Security Inc. This algorithm is immune to differential and linear cryptanalysis. This document illustrates the implementation of RC4 based Pseudo-Random Number Generator (PRNG) for variable key sizes of five to ten bytes.

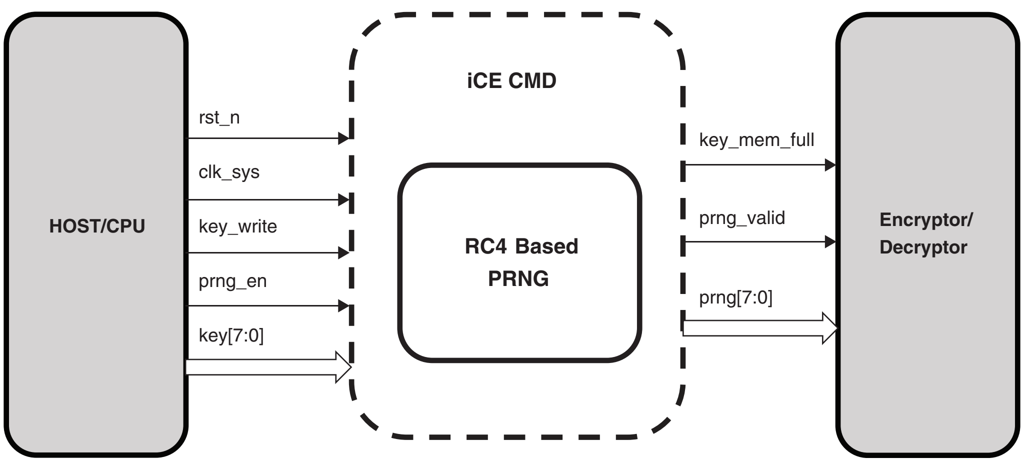
The design is implemented in VHDL. The Lattice iCEcube2™ Place and Route tool integrated with the Synopsys Synplify Pro® synthesis tool is used for the implementation of the design. The design can be targeted to other iCE40™ FPGA product family devices.

Features

- RC4 variable key-size stream-cipher algorithm
- Configurable keys of size five to ten bytes
- Delivers pseudo random numbers as byte streams
- Hardware tested for encryption and decryption
- Useful in stream cipher crypto engines

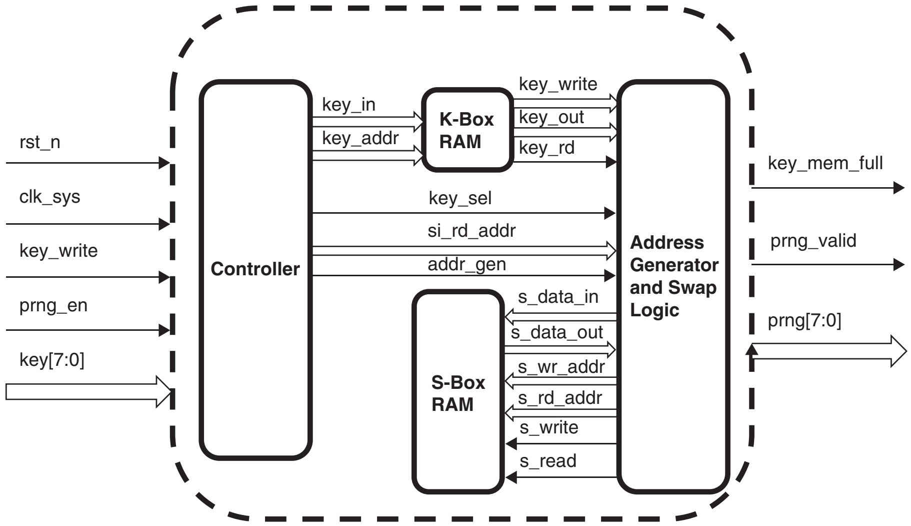
System Block Diagram

Figure 1. System Block Diagram



Functional Description

Figure 2. Functional Block Diagram



RC4 is a variable key-size stream cipher. RC4 generates a pseudo-random stream of bits, which for encryption, is combined with plain text using bitwise exclusive–OR (XOR). Decryption is performed in the same way.

RC4 stream cipher has two phases, the key set up and the key stream generation.

The key permutes a 256 byte array. Once the permutation procedure is completed, the key is never used again. A byte stream is then systematically selected from a 256–byte array and used to encrypt or decrypt data.

RC4 uses a variable-length key from one to 156 bytes. A 256 byte array is used in RC4. In the initialization process, S[i] is i, where i is from 0 to 255. The variable-length key is placed in an array K. The same key is used to do initial permutation of S.

```
// Initialization
for i = 0 to N
S[i] = i;
j = 0;
// key shuffling
for i = 0 to N
j = (j + S[i] + K[i mod l]) mod N; // K is secret Key and l is K size in bytes
Swap(S[i],S[j]);
```

Finally, a single byte value is chosen from the permuted S and some two bytes are swapped in the S.

```
//Stream generation
i= j = 0
i = (i + 1) mod256
j = (j + S[i]) mod256
swap S[i] and S[j]
t = (S[i] + S[j]) mod256
PRNG = S[t] // which is the output PRNG
```

Control FSM controls the process of the key setup and the key stream generation phases.

Key Setup Phase

A low on key_write initiates the key setup phase. When key_write is low, K–Box RAM starts storing the key, Key_mem_full output goes high after writing key of variable lengths into K–Box RAM. S–Box RAM is initialized to 0 to 255 linearly. After the initialization of S–Box RAM and K–Box RAM, the shuffling of data is done for 256 iterations, where a new address is computed for each iteration using the key. Swapping of data takes place in S–Box RAM.

Key Stream Generation Phase

The key stream generation phase is initiated after the completion of the key setup phase, if prng_en is low and key_write is high. This phase also generates a new address without using the key. The shuffling of data is performed the same way as in the key setup phase. The prng [7:0] output is obtained with high on prng_valid output. If prng_en is high and key_write is low, indicating the presence of a new key, the key setup phase is initialized for the new key.

Timing Diagram

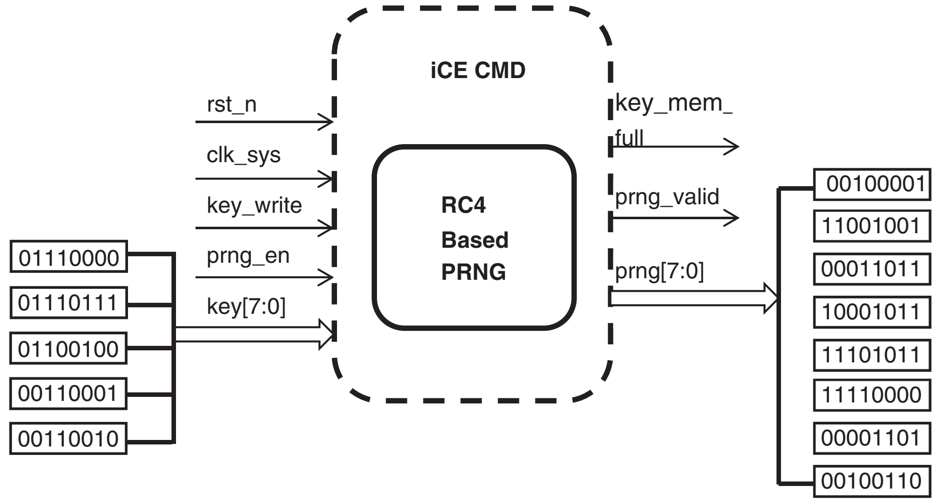
Signal Description

Table 1. Signal Description

Signal	Width	Type	Description
clk_sys	1	Input	System Clock, at 32 MHz
rst_n	1	Input	Active low asynchronous reset
key_write	1	Input	Active low key write enable
prng_en	1	Input	PRNG generation enable signal, generates PRNG byte streams when held low
key	8	Input	Key input, Multi-byte write supported with a low key_write for multiple cycles
key_mem_full	1	Output	Key Memory full indicator. A high on this indicates that the key memory is full
prng	8	Output	PRNG output
prng_valid	1	Output	PRNG output valid. A high on this indicates a valid PRNG output

Operation Sequence

Figure 3. Operation Sequence



Note that rst_n must be held low initially to bring up the design in a correct operating state.

Timing Diagram

Figure 4. Timing Diagram of Key Writing

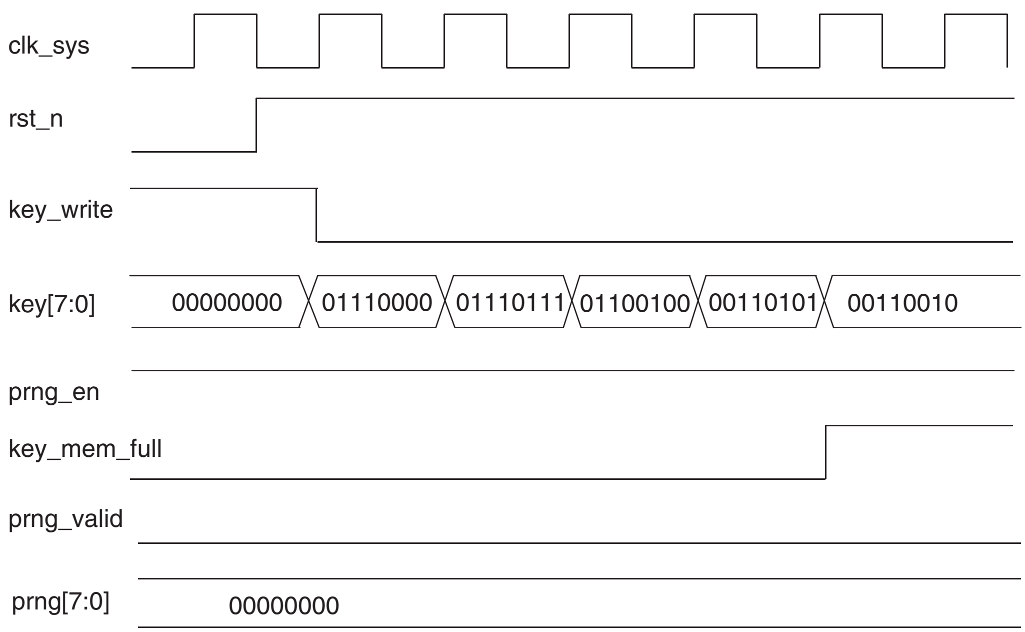
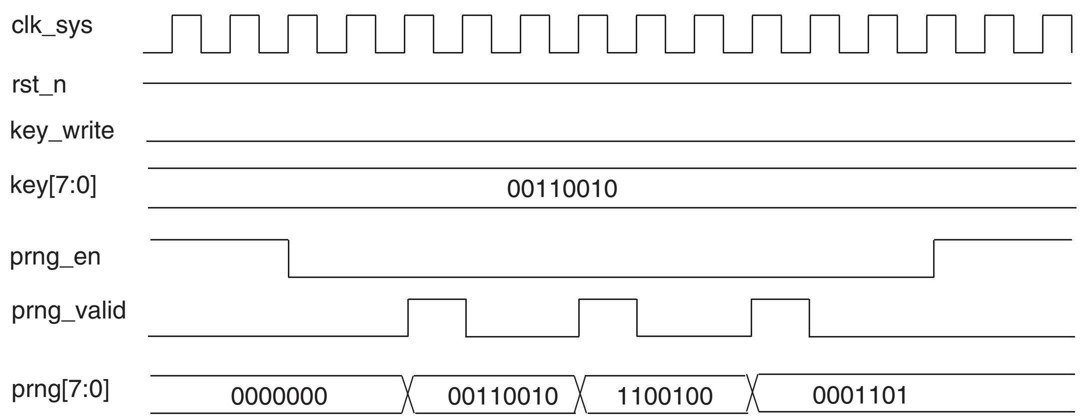


Figure 5. Timing Diagram of Valid prng Output



Simulation Waveforms

Figure 6. Simulation Waveforms for Key Writing

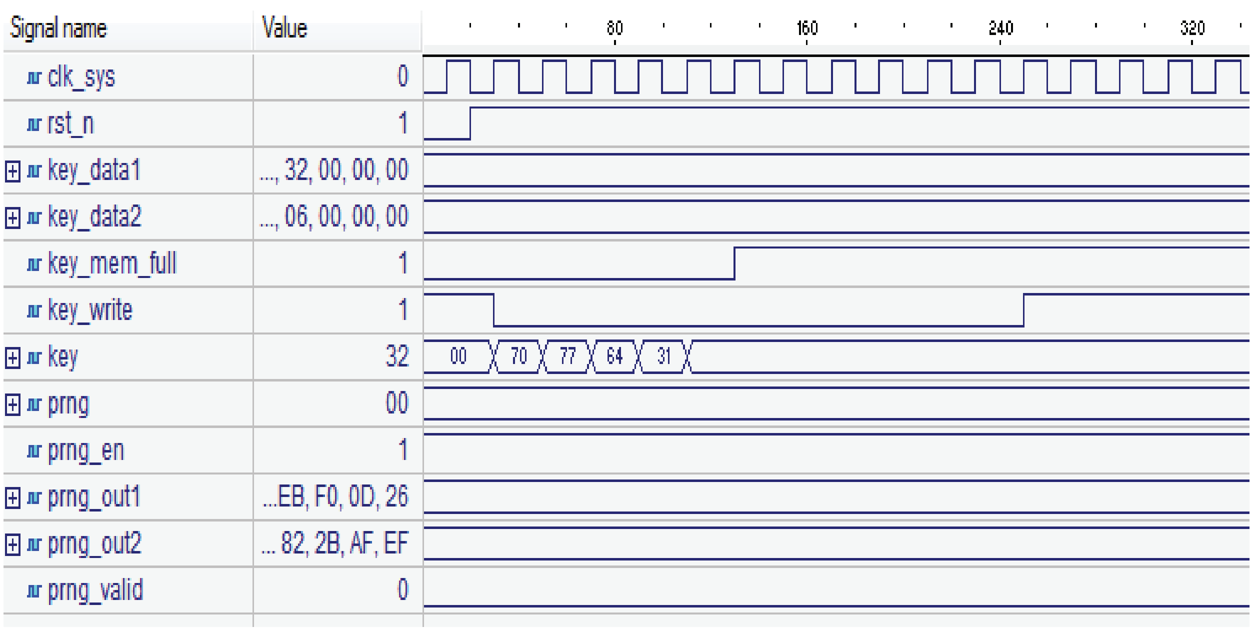
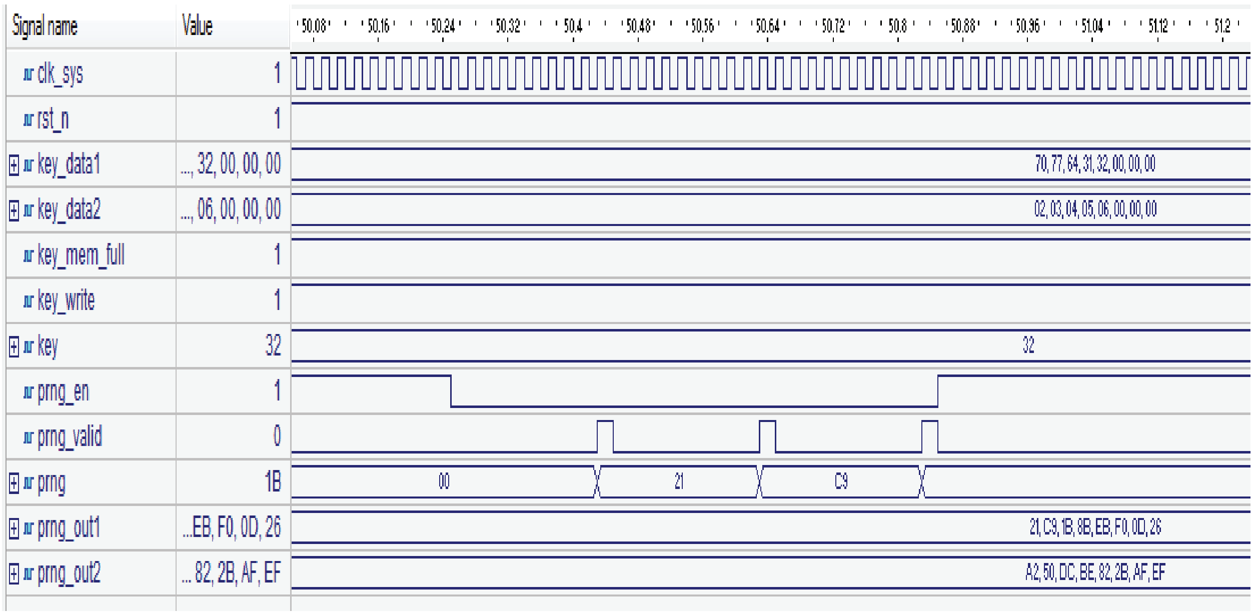


Figure 7. Simulation Waveforms for Valid prng Output



Implementation

This design is implemented in VHDL. When using this design in a different device, density, speed or grade, performance and utilization may vary.

Table 2. Performance and Resource Utilization

Family	Language	Utilization (LUTs)	f_{MAX} (MHz)	I/Os	Architectural Resources
iCE40[1]	VHDL	201	>50	22	(46/160)PLBs

1. Performance and utilization characteristics are generated using iCE40LP1K-CM121 with iCEcube2 design software.

References

- iCE40 Family Handbook

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2013	01.0	Initial release.