High Precision Discrete Gaussian Sampling on FPGAs

Selected Areas in Cryptography 2013

Sujoy Sinha Roy, Frederik Vercauteren and Ingrid Verbauwhede

ESAT/COSIC and iMinds, KU Leuven

Outline of Talk

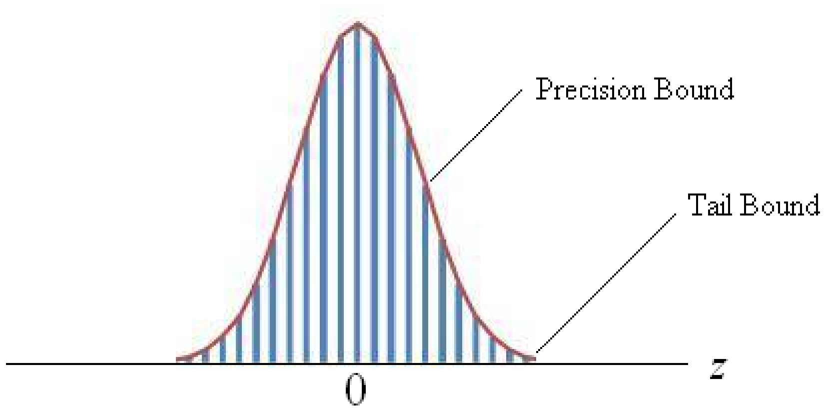
- Introduction
- Implementation of discrete Gaussian sampling using Knuth-Yao Random Walk
  - Basics of Knuth-Yao sampling
  - Implementation of Knuth-Yao random walk using counters
  - Space optimization for Probabilities
  - Hardware architecture
  - Results

Discrete Gaussian Sampling

- Discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ over $\mathbb{Z}$ with mean 0 and standard deviation σ

$$Pr(E=z) = \frac{1}{S} e^{-z^2/2\sigma^2} \quad where\ S = 1 + 2\sum_{z=1}^{\infty} e^{-z^2/2\sigma^2}$$

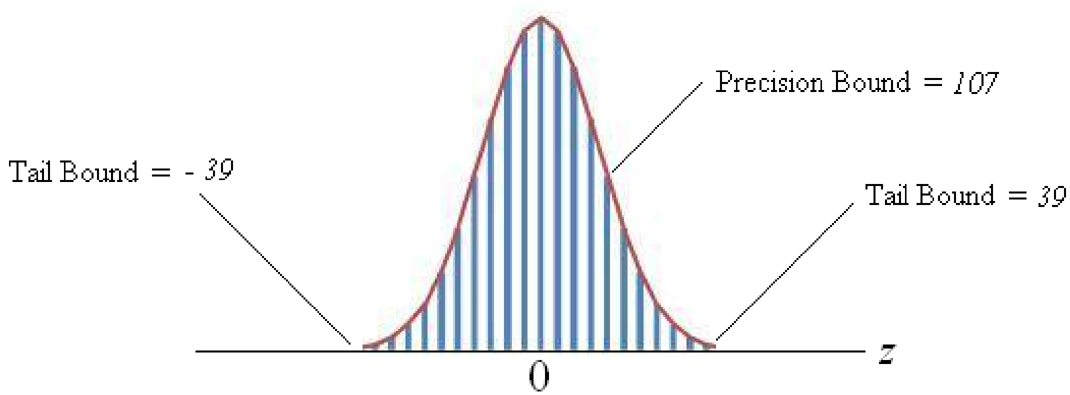
- Tail is infinitely long
- Probabilities have infinite precision



Discrete Gaussian Sampling : Tail/Precision Bounds

- Provable Security :
  - Negligible statistical distance from true Gaussian distribution : 2^{-90}
  - For standard LWE parameter set
    - Tail bound : practically 39

m	s	σ	Tail	Precision
256	8.35	3.33	84	106
320	8.00	3.192	86	106
512	8.01	3.195	101	107



Sampling Methods

- Commonly used methods
  - Rejection sampling
  - Inversion sampling

- Large number of random bits are required to maintain high precision

- Slow on resource-constrained platforms

Knuth-Yao Sampling

- Random-walk model
- Requires near-optimal number of random bits
- Example : Let a sample space $S = \{0, 1, 2\}$

$$
\begin{aligned}
p_0 &= 0.01110 \\
p_1 &= 0.01101 \\
p_2 &= 0.00101
\end{aligned}
$$

- Probability matrix

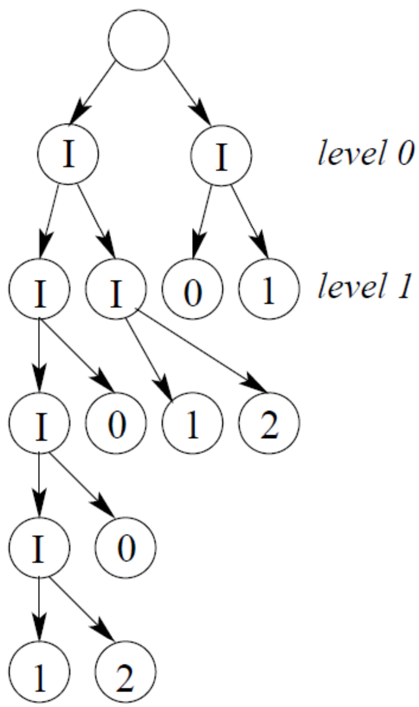
column 0 (↓)

row 0 (→)

$$
P_{mat} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}
$$

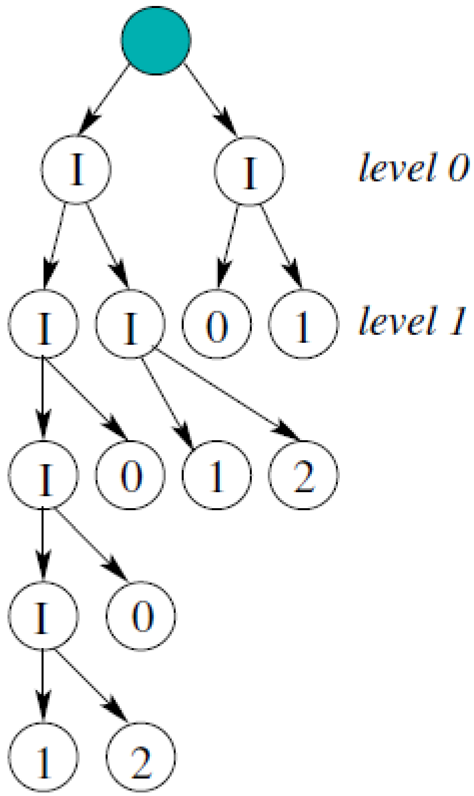
Knuth-Yao Sampling

- Discrete Distribution Generating (DDG) tree is formed
  - Binary tree corresponding to P_{mat}

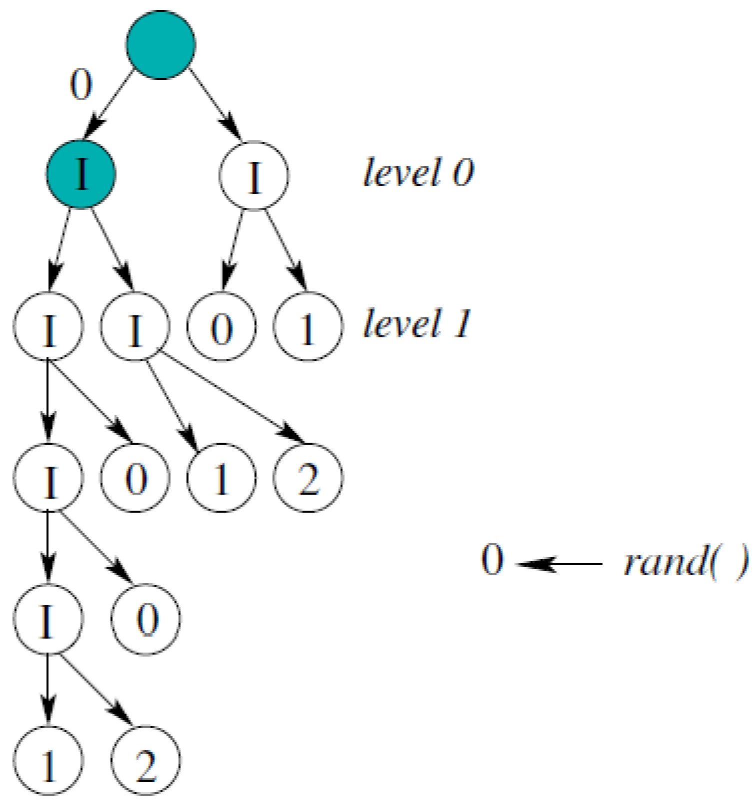


$$P_{mat} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

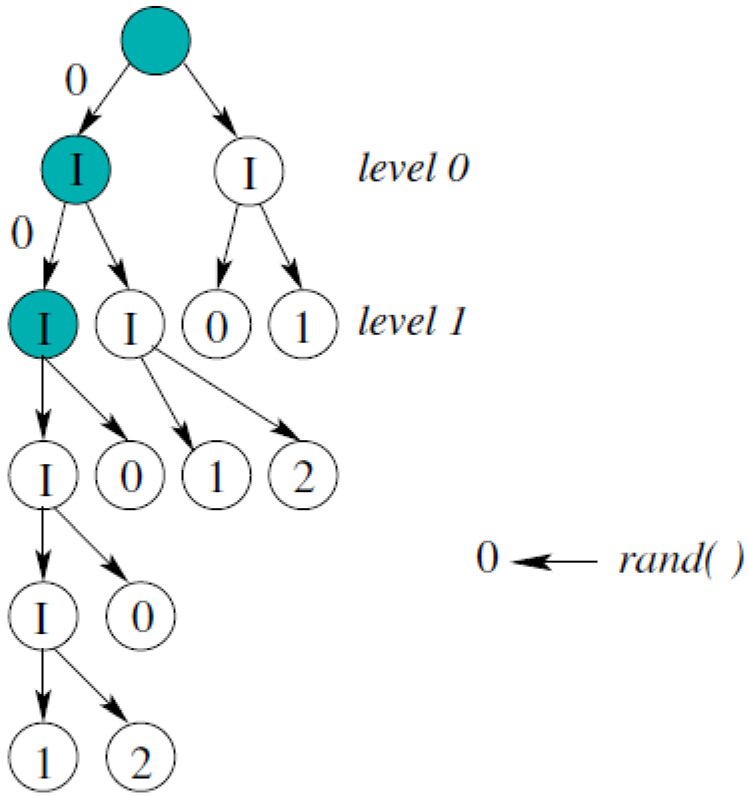
Knuth-Yao Sampling : Random Walk



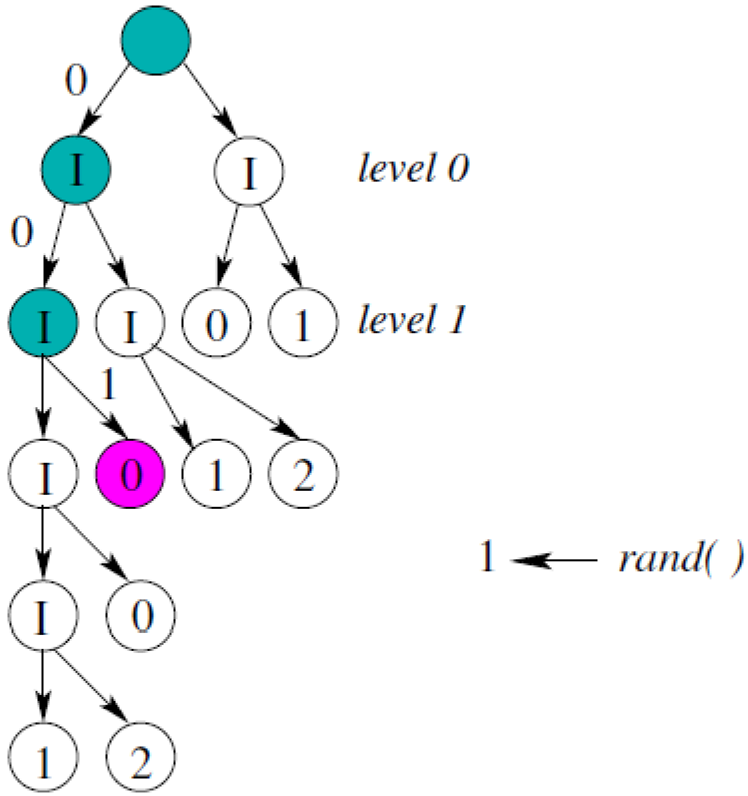
Knuth-Yao Sampling : Random Walk



Knuth-Yao Sampling : Random Walk

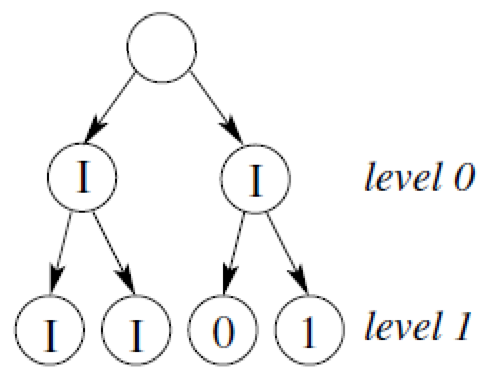


Knuth-Yao Sampling : Random Walk



Knuth-Yao Sampling : Implementation

- Any level of the DDG tree can be constructed from previous level using probability matrix

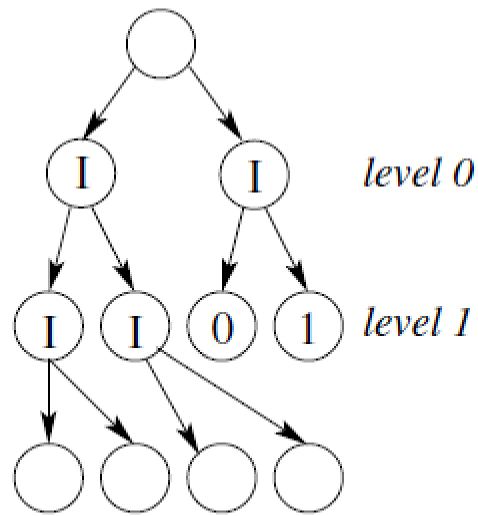


$$P_{mat} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Knuth-Yao Sampling : Implementation

- Any level of the DDG tree can be constructed from previous level using probability matrix

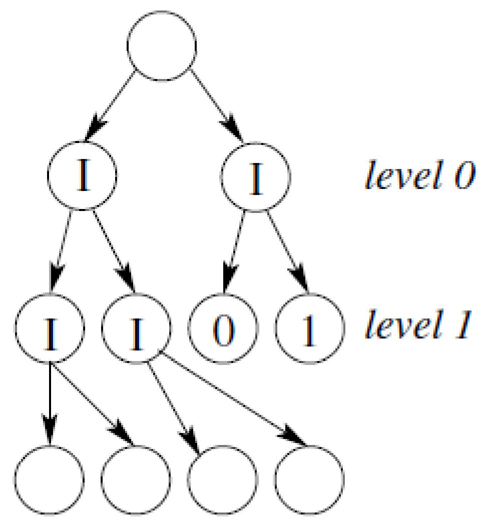
$$P_{mat} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$



Knuth-Yao Sampling : Implementation

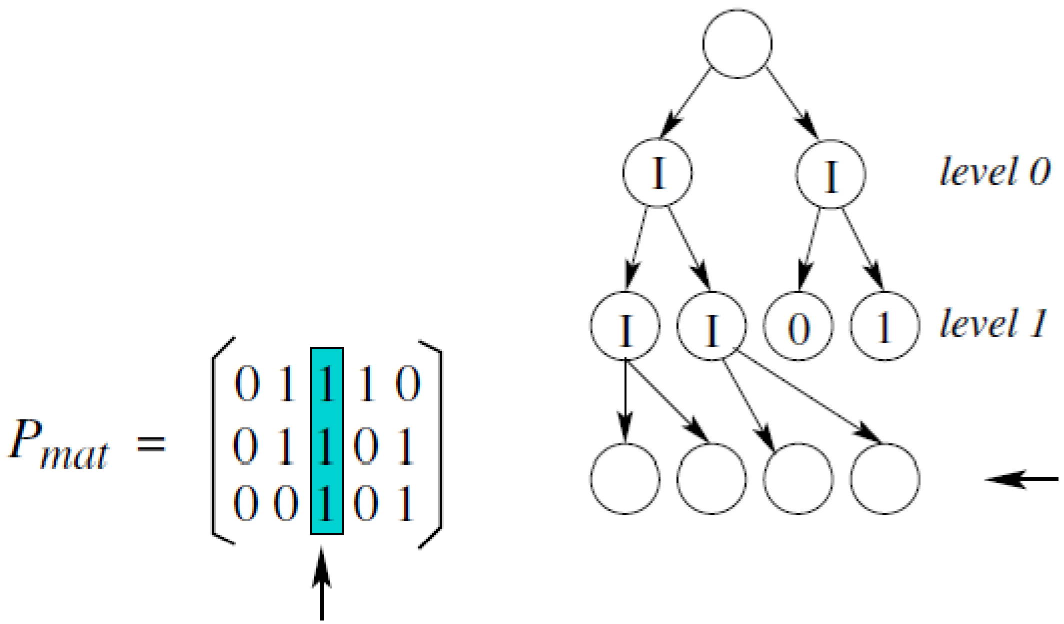
- Any level of the DDG tree can be constructed from previous level using probability matrix

$$P_{mat} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$



Knuth-Yao Sampling : Implementation

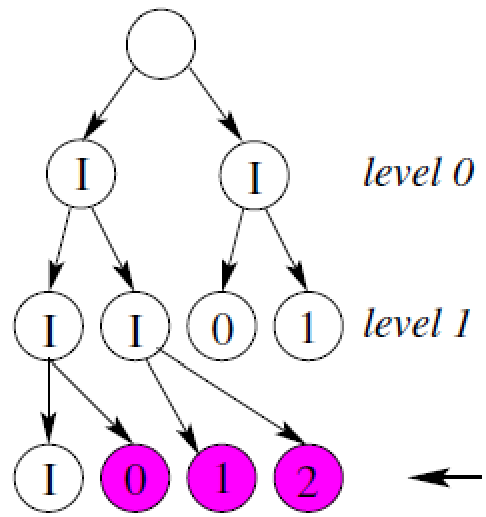
- Any level of the DDG tree can be constructed from previous level using probability matrix



Knuth-Yao Sampling : Implementation

- Any level of the DDG tree can be constructed from previous level using probability matrix

$$P_{mat} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$



Knuth-Yao Sampling : Two Important Points

- Knuth-Yao random walk

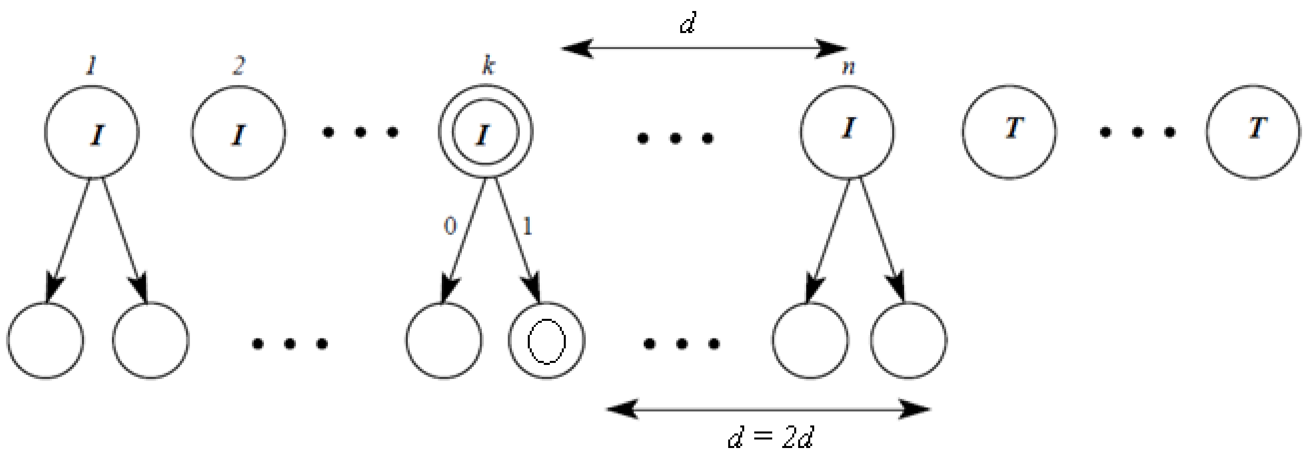
- Storage for Probability Matrix

Knuth-Yao Sampling

Random Walk using Counters

Knuth-Yao Sampling : using Counters

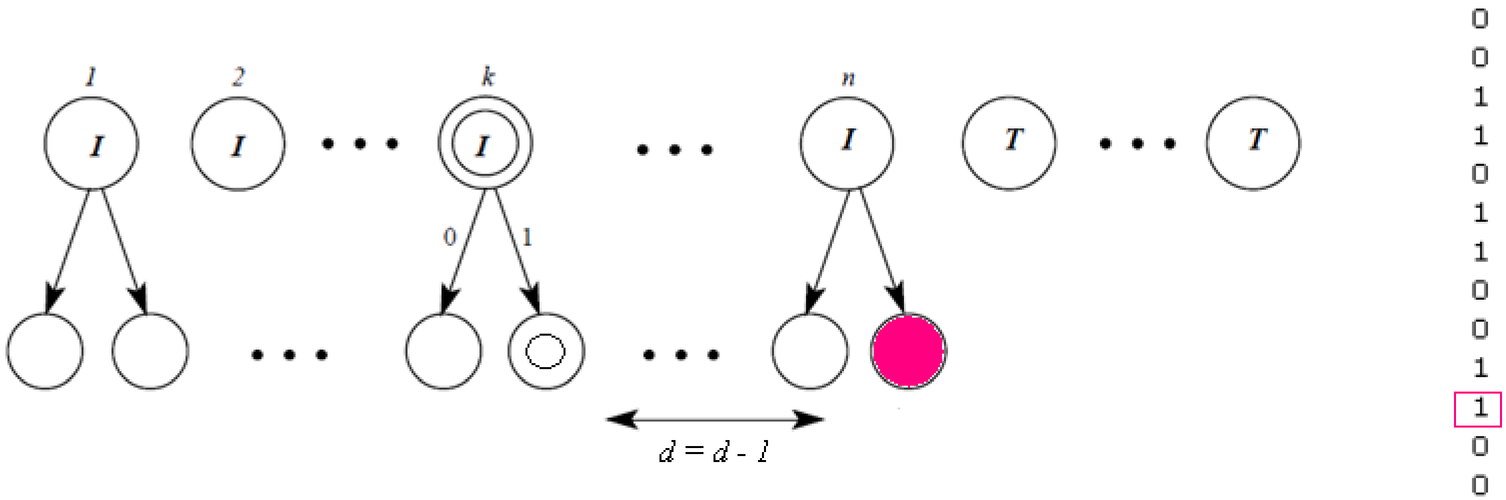
- Construction of *i-th* level during sampling : Counter *d* for distance



0
0
1
1
0
1
1
0
0
1
1
0
0

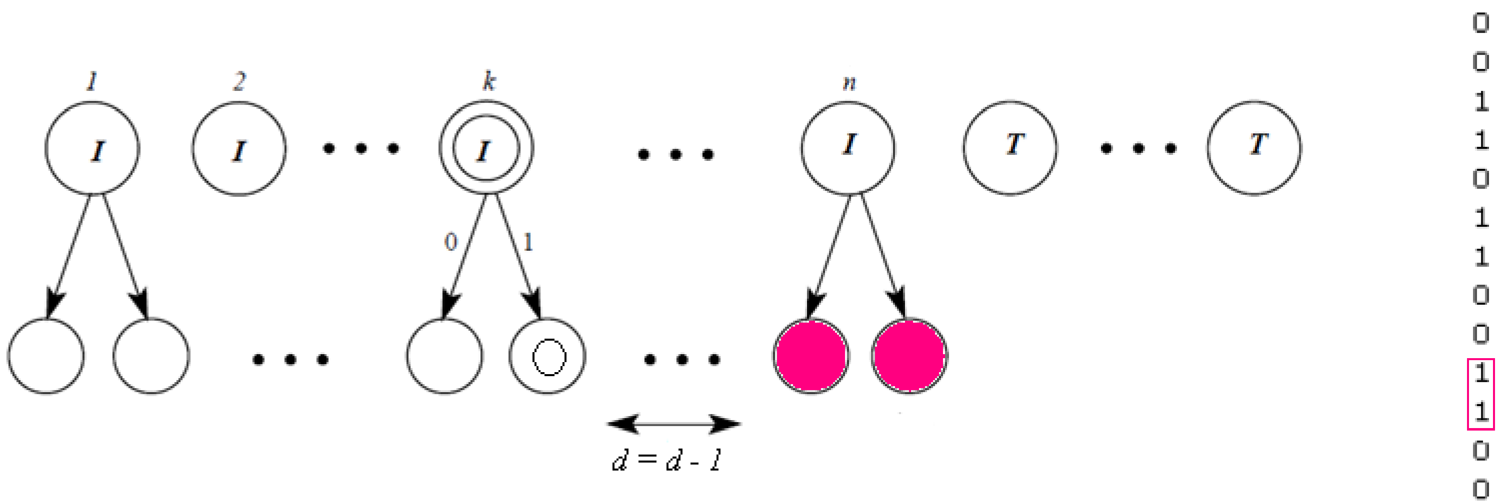
Knuth-Yao Sampling : using Counters

- Construction of *i-th* level during sampling : Counter *d* for distance



Knuth-Yao Sampling : using Counters

- Construction of *i-th* level during sampling : Counter *d* for distance



- When $d < 0$ for the first time, the visited node is a terminal node
- We need counters for *d* and row-number

Space Optimization for Probability Matrix

Probability Matrix : 107-bit precision, 39-tail-bound s=8.01


```
000111111111010111000101110101010010110001101100101010100000011100000011111111101011100010111010101001011000
001111001101110110011011001101000010101001100000001111000011011110111011100111100110000000000111001011101111
001101001000110011101100011010110000000110111001000111100011011110011001011101001110011110001000010011000010
001010010010001110000011001110001110000011001101001001010101100011110110010000110111101100101010111111111111
000111010011001101100110100000100001001100011010000010101010101000011100111011000010001001000010111111010000
000100101100101100100011010010101110101011111110000101100111100010100101010000000111110111110000100011100010
000010101111011110010010001110000001101100100100100010101011010101111100000110111011011011110111100110100
000001011100110110001001011000110110001001001001001100001101010110101100011101111001110101010010100011001010
000000101100100010110010101101111110011110011011110010111010101101110101110111011010011011011000100111101
000000010011011000000110100010100001000101101000111001010000101101110001101110111101101011101101010011010
000000001111010010001111110111110000011010100100111000000101100110000101011111101010111111110010110001000
000000000101011101110110010011101101001011011010101011101101000000100000110001110010101111111101011111100110
000000000000111000101110001100111100101101110110110011001000011011101111000101010100100101010001001111001000
000000000000010000101011010101100000111001100101100000100110110111111010010101100111101101110100001010111110
000000000000000100011100100010011110010000000101101000001101010101111110011010011000111111011011100011111000
000000000000000001000100110001110011100100010100110101111110011010010110000100001000011111000100101001000000
000000000000000000001111000100101111010101111000010111011100111010110001001011001001010100111001101101000000
000000000000000000000101111111011011101100000011110110110011000010111001110110101011111010100110111100010000
000000000000000000000000100010100010110101110110100011001101110000000111001000101101111000010000011001000010
000000000000000000000000001011010010011001011001000000101001001111110101100010000001111111011000000100000110
000000000000000000000000000000110101100000001100100100001110011001001110011000100010001000000101100111111011
000000000000000000000000000000001110010111111001101001101100100001010011011101010001001111000110000001110000
000000000000000000000000000000000001110000000001111110110100111101101001100111010100100011100101001111110000
000000000000000000000000000000000000001100010111101111110011100110010110100011010001100110011011011001100100
000000000000000000000000000000000000000010011110100010110110000111111111001000010000111101011001100000000
000000000000000000000000000000000000000000011100110010010101000000101001010100100100011110110001100111110
000000000000000000000000000000000000000000000100101111010011001101011000011001110011001101000110011000100
000000000000000000000000000000000000000000000001011010100011000011101110110000100100010100000001110100
000000000000000000000000000000000000000000000000011000100000101011001000001001111110110000010011101000
000000000000000000000000000000000000000000000000000110000001000000111011100110111010001000001011100
000000000000000000000000000000000000000000000000000000010101011010111010101101000101001101010101100
000000000000000000000000000000000000000000000000000000000010001010010100000100101001011011111100100
000000000000000000000000000000000000000000000000000000000000110010100111001000010011010110010100
000000000000000000000000000000000000000000000000000000000000001000011001010101111011101011100
000000000000000000000000000000000000000000000000000000000000000010100001101001100011110000
000000000000000000000000000000000000000000000000000000000000000000101100000101111011100
000000000000000000000000000000000000000000000000000000000000000000000010101110011111000
000000000000000000000000000000000000000000000000000000000000000000000000100111001000
000000000000000000000000000000000000000000000000000000000000000000000000000000001110
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Probability Matrix : Column-wise Optimization

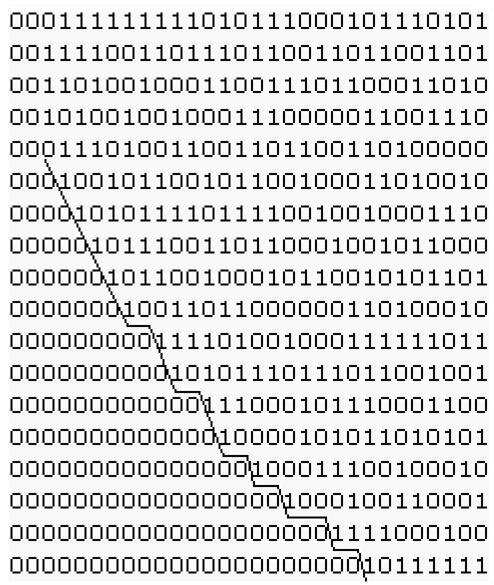
- Probability matrix is stored in ROM in a column-wise manner
  - Zeros present in bottom of a column are not stored
  - Lengths of columns are also stored

```
000111111111010111000101110101
001111001101110110011011001101
001101001000110011101100011010
001010010010001110000011001110
000111010011001101100110100000
000100101100101100100011010010
000010101111011110010010001110
000001011100110110001001011000
000000101100100010110010101101
000000010011011000000110100010
000000000111101001000111111011
000000000010101110111011001001
000000000000111000101110001100
000000000000010000101011010101
000000000000000100011100100010
000000000000000001000100110001
000000000000000000001111000100
000000000000000000000010111111
```

Probability Matrix : Column-wise Optimization

- Observation
  - ➢ Difference in length is 1 for most consecutive columns

- We consider one-step difference in column length
  - ➢ One bit is required per differential column-length
    - 1 for increment
    - 0 for no-increment

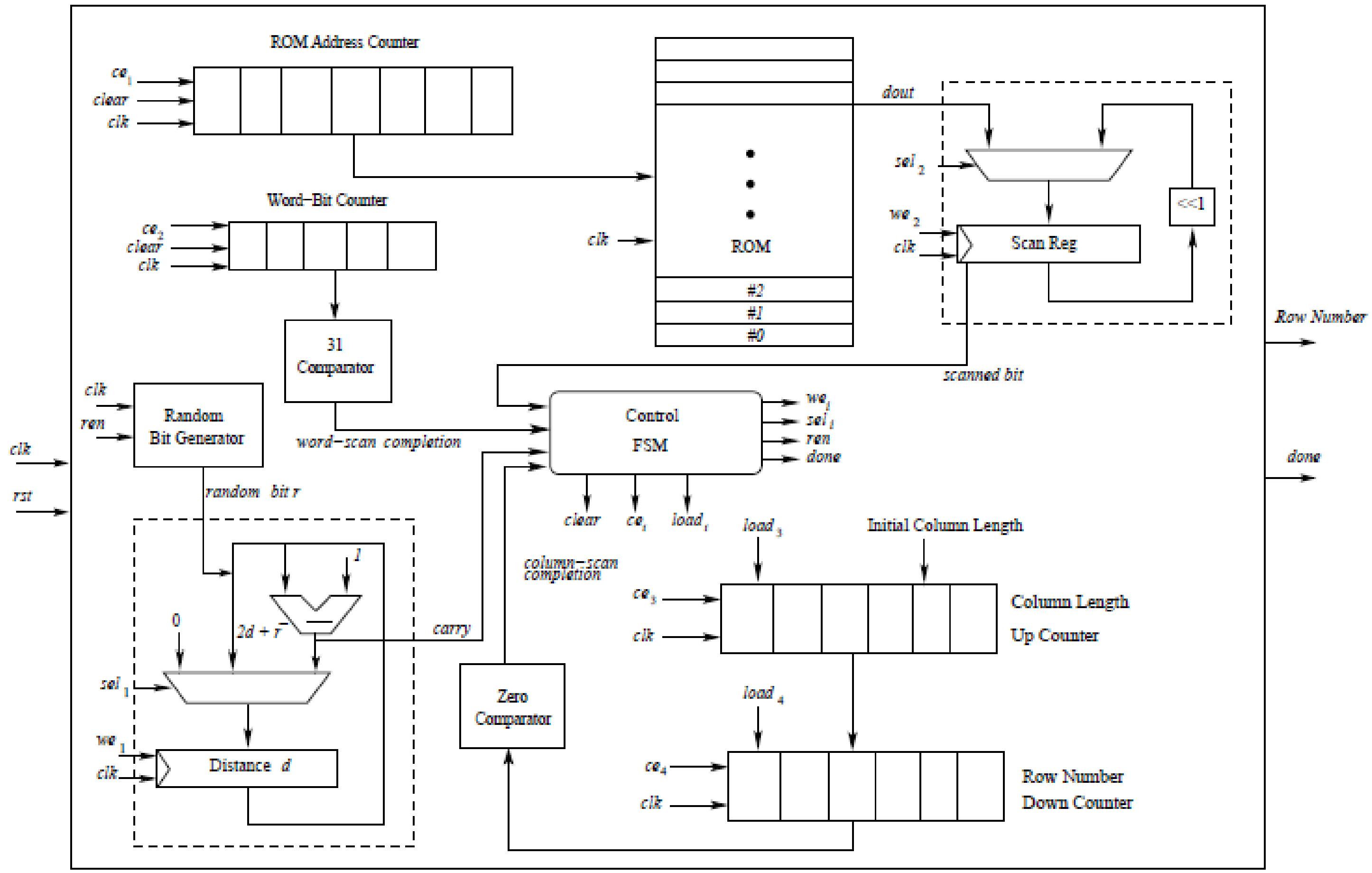


Hardware Architecture

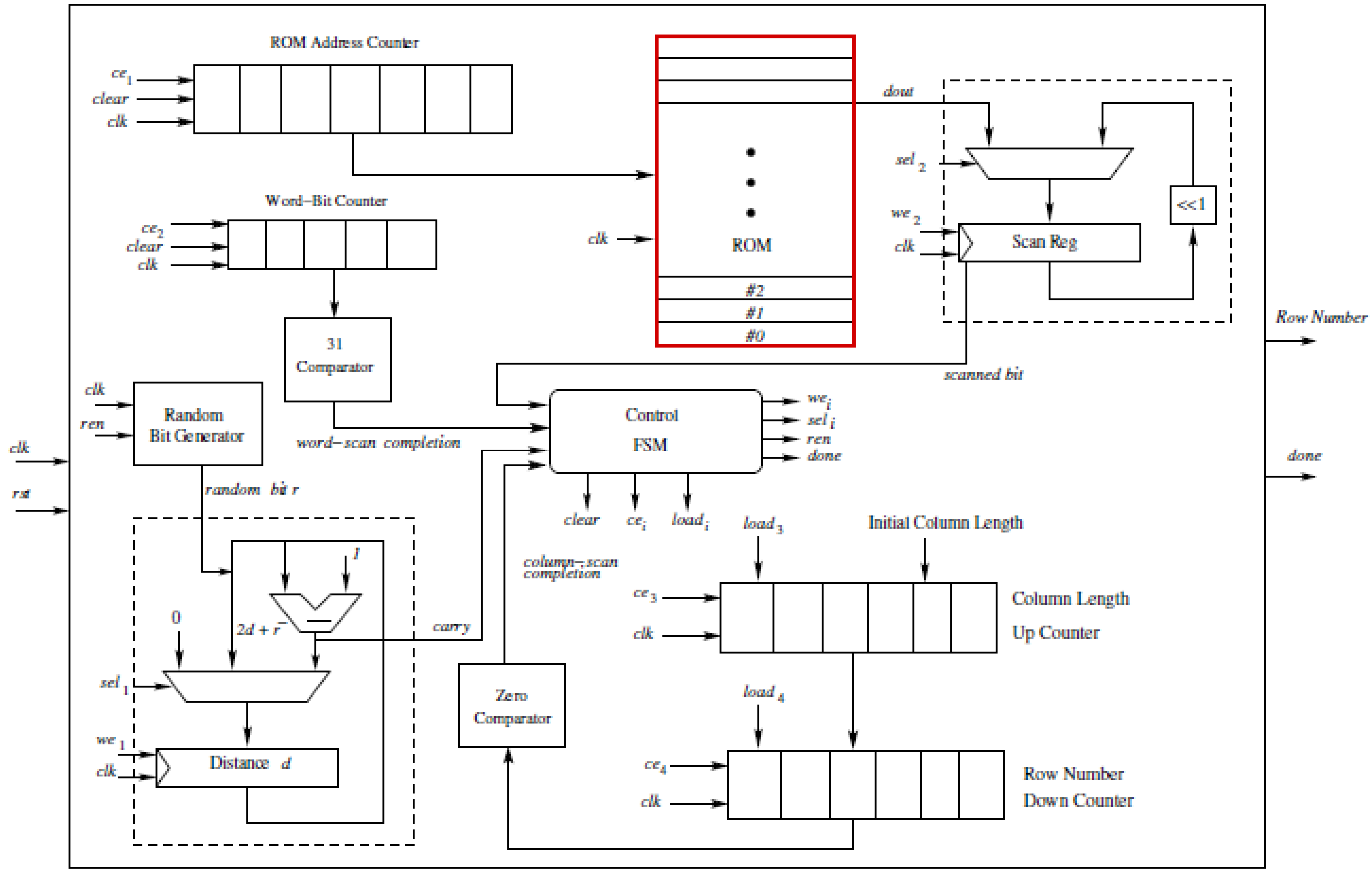
Hardware Architecture

- Components required
  - ROM for storing probability matrix
  - Counters for *d*, *row* and *column* during Knuth-Yao sampling
  - Comparators for checking terminal conditions
  - Shift-register for scanning columns

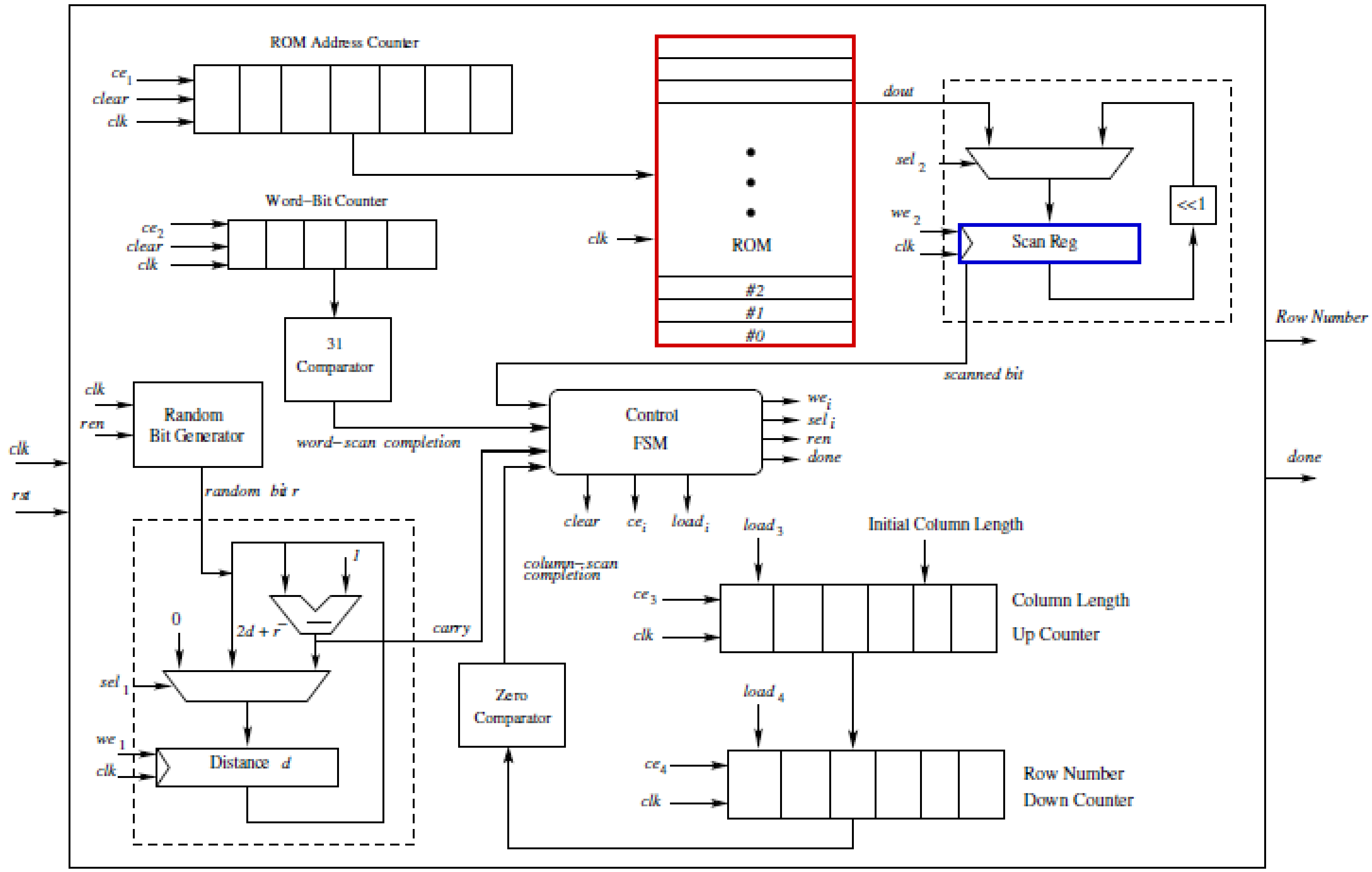
Hardware Architecture



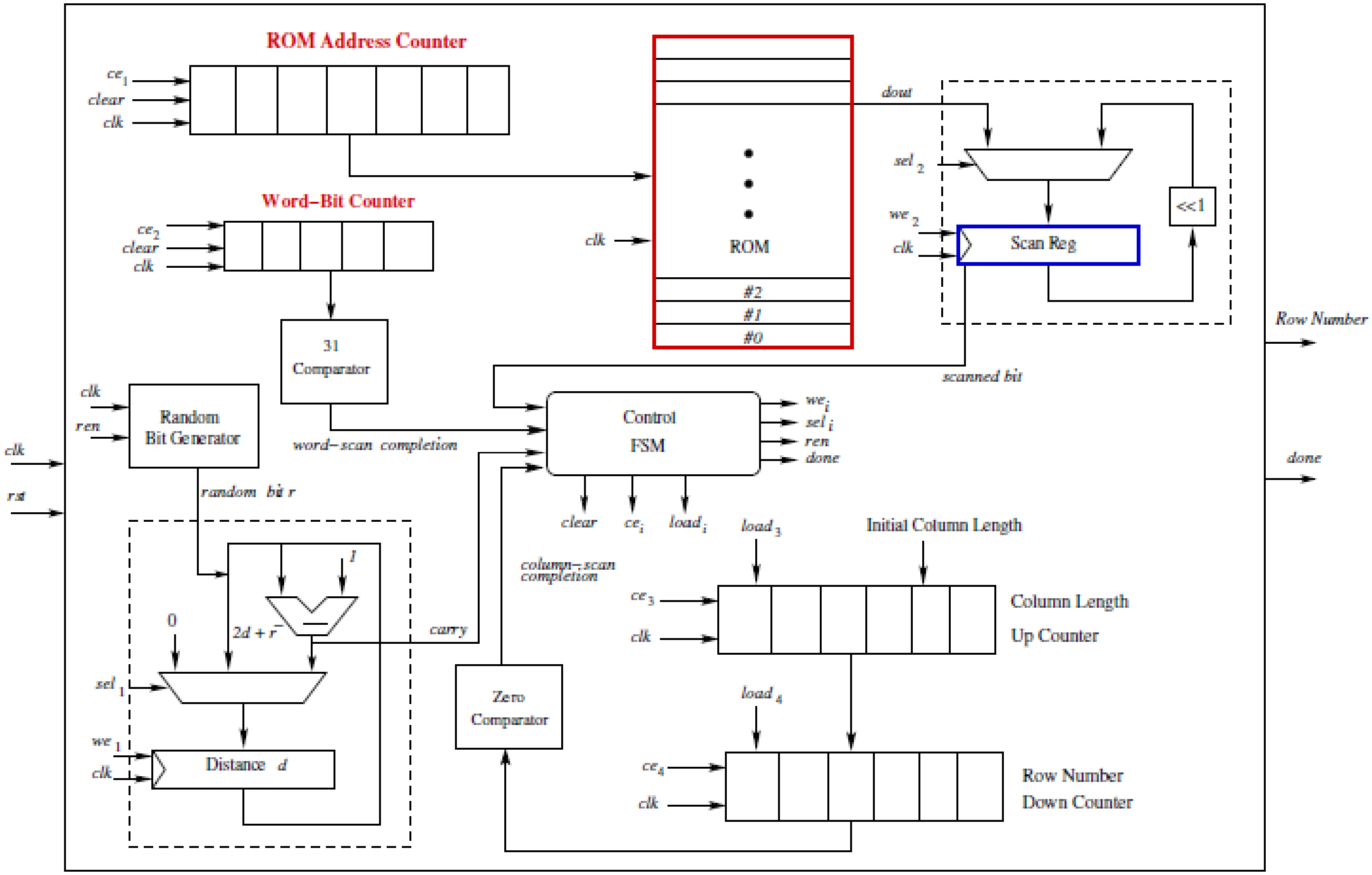
Hardware Architecture



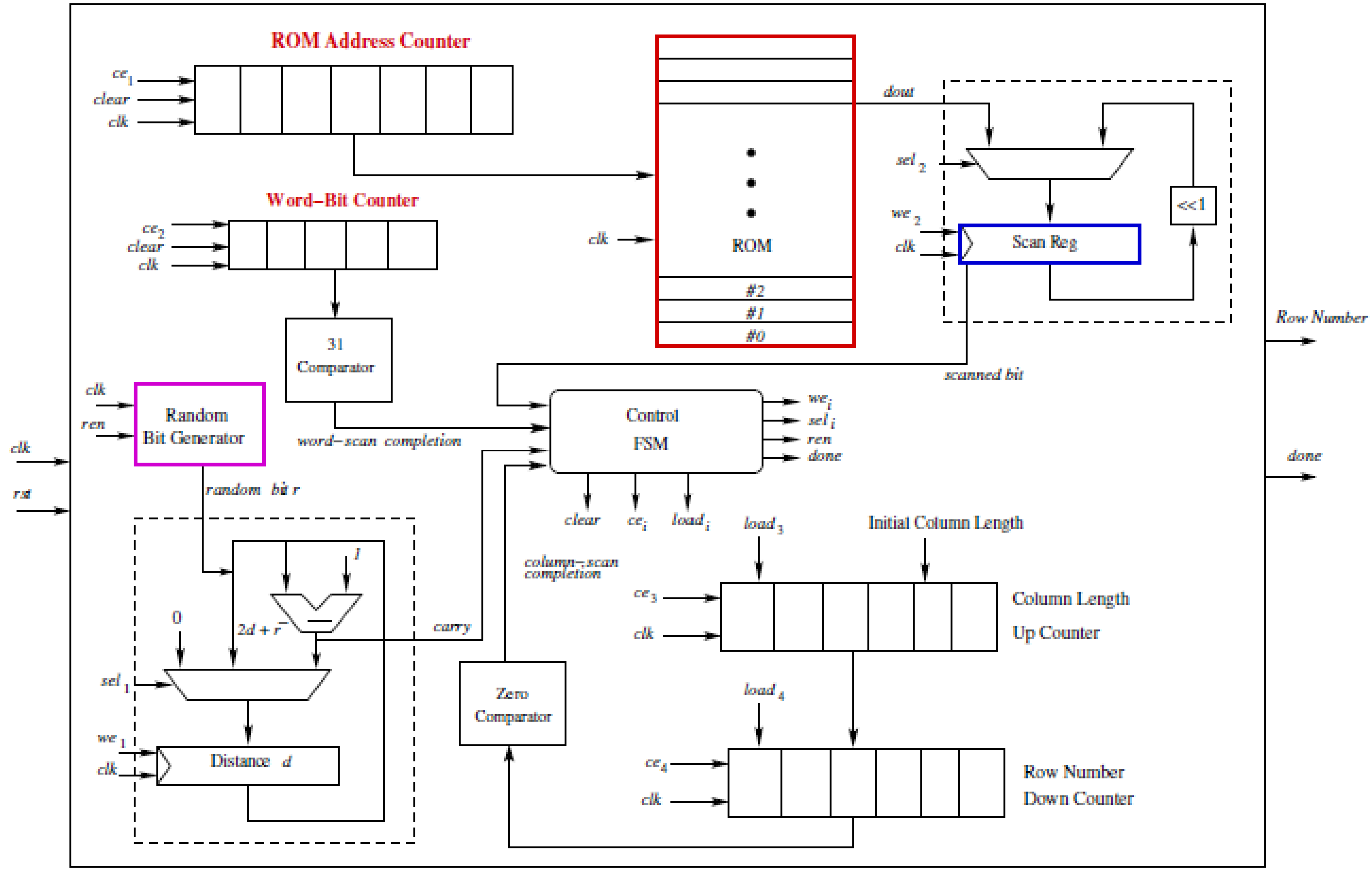
Hardware Architecture



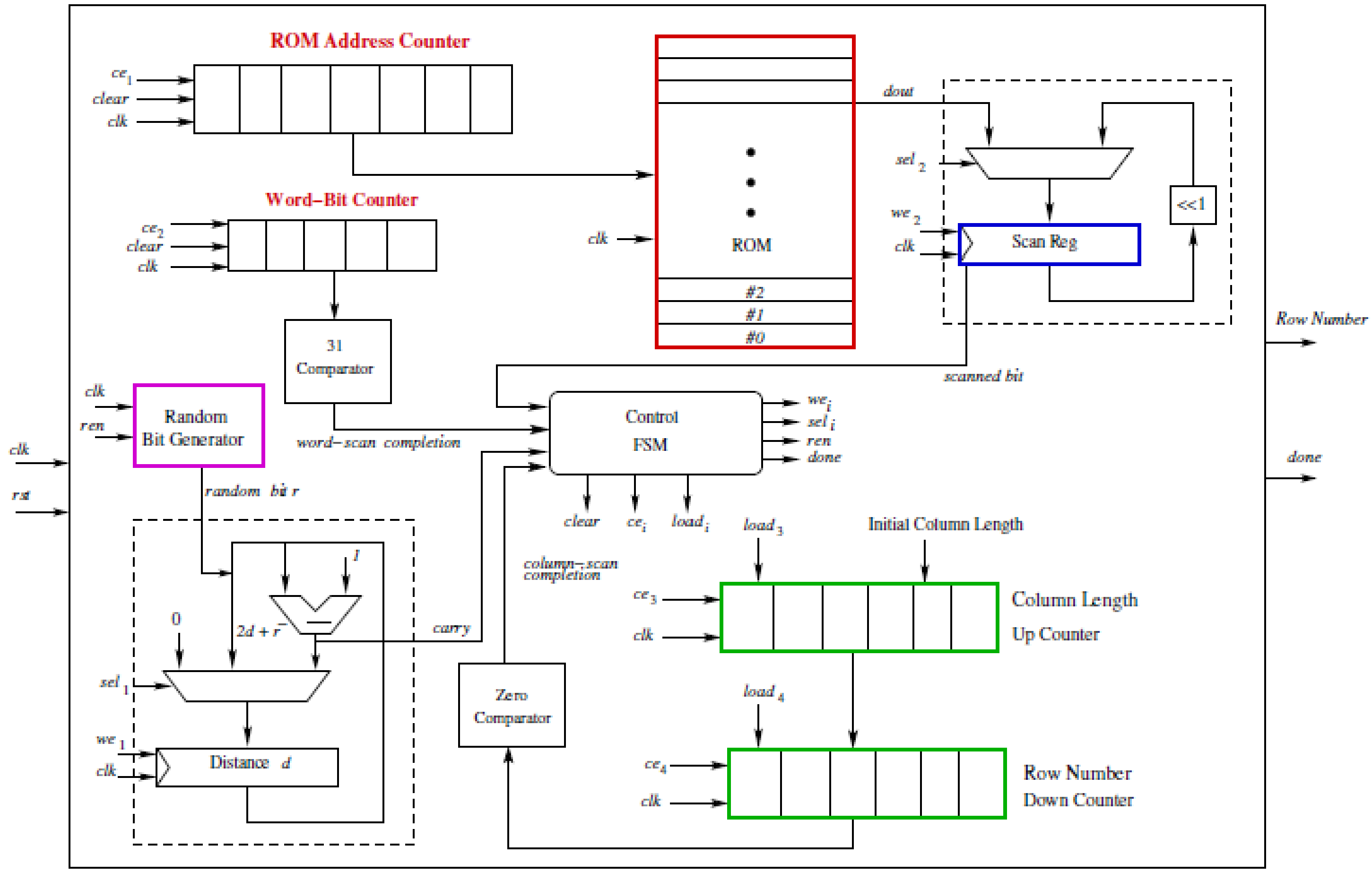
Hardware Architecture



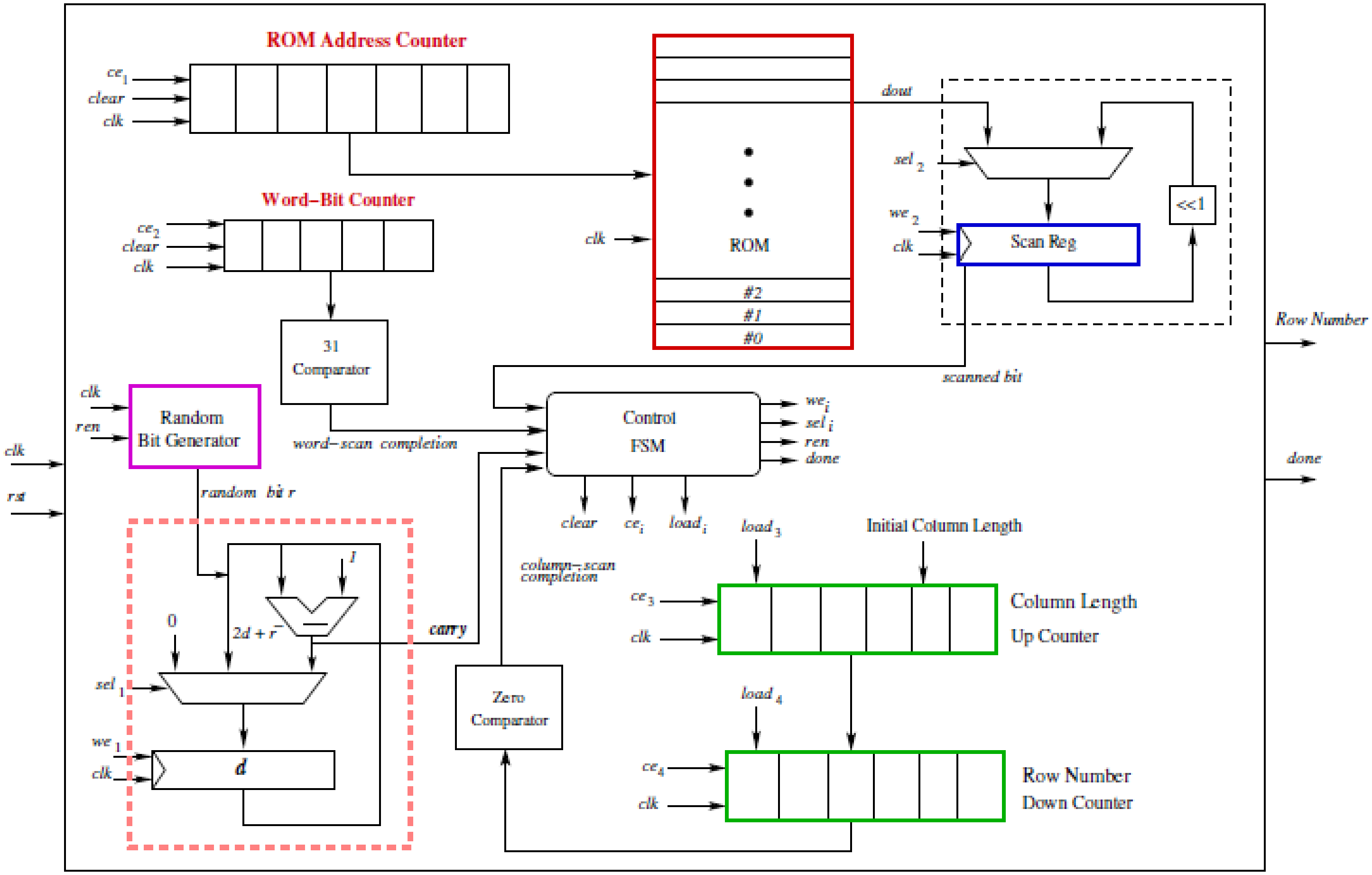
Hardware Architecture



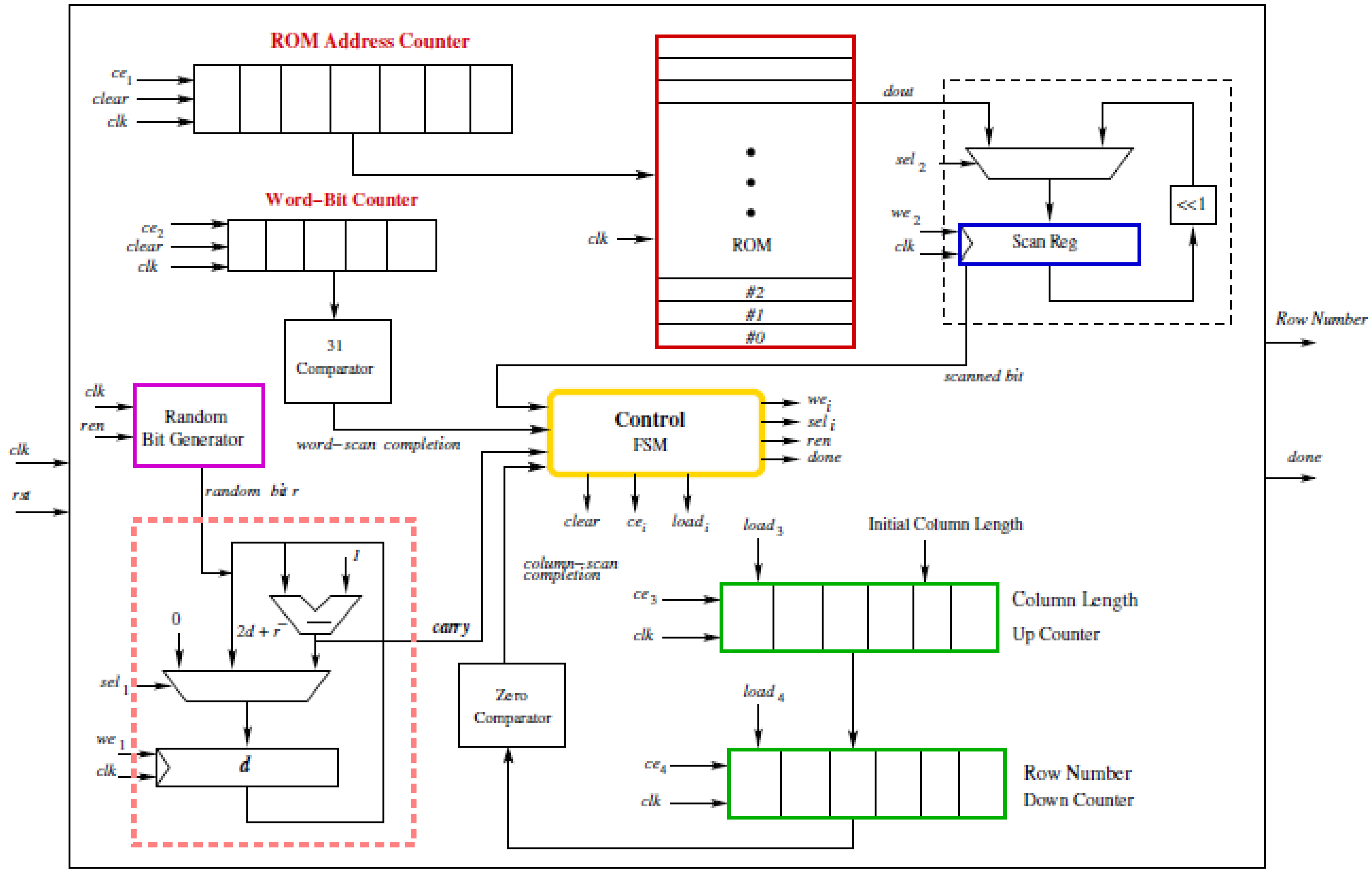
Hardware Architecture



Hardware Architecture



Hardware Architecture



Hardware Architecture : Speeding-Up

- Scanning of the column bits is the most time consuming operation.
- Hardware => Parallelism
  - Window-based scanning of column bits
  - Reduces scanning time
  - Marginal increase in area

Experimental Results

Architecture	FFs	Slices		LUTs		Delay (ns)	Clock Cycles
		Core	ROM	Core	ROM		
Basic	66	30	17	76	64	3	17
Window	69	36	17	85	64	3.3	16

Performance of the discrete Gaussian sampler on xc5vlx30

- Storage for Probability matrix
  - 32-by-96 ROM
  - Results do not include the Random Bit Generator
  - Window method provides acceleration for long random walks
  - Timing Analysis is present in the paper

Conclusion & Future Work

- Hardware implementation of high-precision discrete Gaussian sampler.
  - Efficient implementation for small standard deviation
  - Storage is an issue for large standard deviation

- Implementation of LWE cryptosystem
  - Polynomial multiplier and discrete Gaussian sampler in pipeline
  - Sampler is slower than polynomial multiplier
  - Parallelization of sampler cores is possible

Thank You