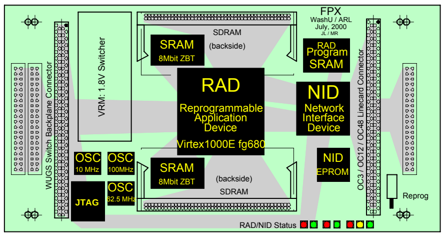
CprE / ComS 583 Reconfigurable Computing

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

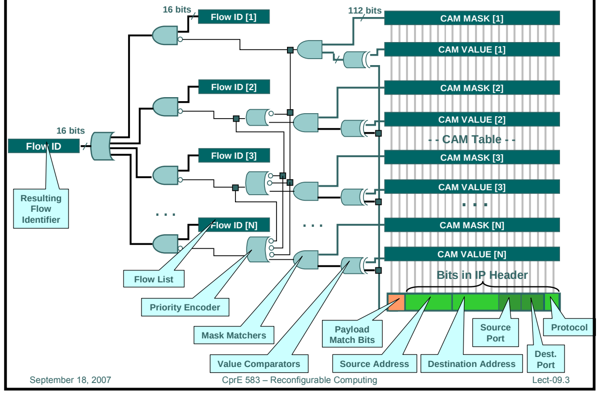
Lecture #9 – Applications II

Recap – FPGA-Based Router (FPX)

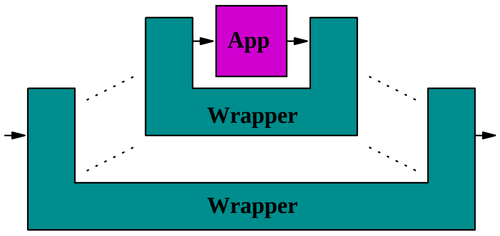


- FPX module contains two FPGAs
- NID – network interface device
  - Performs data queuing
- RAD – reprogrammable application device
  - Specialized control sequences

Recap – Classification Architecture



Recap – The Wrapper Concept

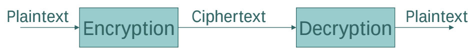


Outline

- Recap
- Cryptography on FPGA Platforms
  - Introduction to cryptography
  - Motivation
- Applications
  - Secure hashing
  - Symmetric-key cryptography
  - Random number generation

Introduction to Cryptography

- *Encryption* is the process of encoding a message such that its meaning is not obvious
- *Decryption* is the reverse process, i.e., transforming an encrypted message to its original form



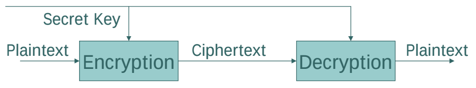
- We denote plaintext by P and ciphertext by C
- C = E(P), P = D(C) and P = D(E(P)), where E() is the encryption function (algorithm) and D() the decryption function

Terminology

- *Encrypt*, *encode*, *encipher* are interchangeable in the context of cryptography
- Same with *decrypt*, *decode*, and *decipher*
- *Cryptographer* – goal is to use encryption to conceal information
- *Cryptanalyst* – goal is to break the encryption
- *Cryptologist* – researches into both encryption and decryption (both cryptography and cryptanalysis)
- An encryption algorithm is *breakable* if given enough time/memory a cryptanalyst can determine the algorithm
  - Algorithm in this context includes the key
  - Is all encryption breakable?

Kerckhoff's Principle

- How do you prevent an eavesdropper from computing P, given C?
  - Keep the encryption algorithm E() secret
    - Is this a good idea?
  - Choose E() (and corresponding D()) from a large collection, based on secret key
    - Kerckhoff's principle: assume that the potential cryptanalyst knows everything but the key



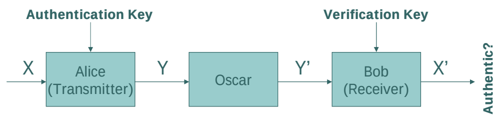
C = E(K, P) and P = D(K, C)

Motivation

- Cryptography is a powerful tool for protecting systems against many types of security threats
- Cryptographic functionality is needed for almost every type of computing platform:
  - From embedded devices to parallel machines
  - Wide range of area and performance requirements
- FPGA technology has become a popular target for implementing cryptographic ciphers
  - Hardware can greatly accelerate the performance of the individual operations required
  - More effective development process than that for ASICs (faster, cheaper)
  - Reconfigurable nature offers additional advantages (algorithmic agility, upload, modification)

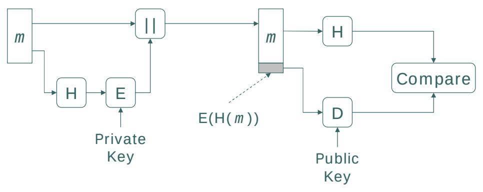
Application – Authentication Codes

- Authentication codes provide assurance that message has not been tampered with and has indeed originated from a specific source
  - Independent of encryption
- *Impersonation Attack*: Oscar introduces a message into the channel, hoping to have it accepted as authentic by Bob
- *Substitution Attack*: Oscar observes a message *Y'* in the channel which he intercepts and replaces by another message *Z'* hoping to have it accepted as authentic by Bob

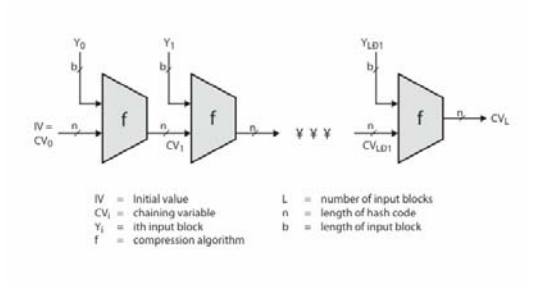


Signing With Message Digests

- A message digest (or hash) function is a one-way function which produces a fixed length vector of an input block *x* of arbitrary length
  - A fixed length "fingerprint" of a message
- Instead of signing message, sign the message digest



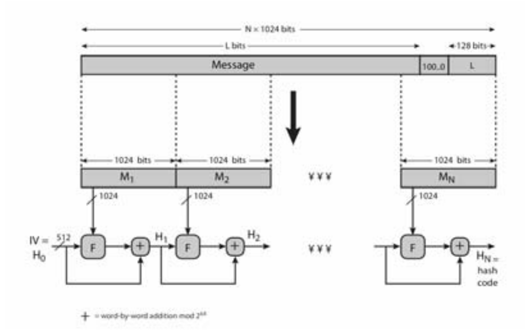
Hash Algorithm Structure



Secure Hash Algorithm (SHA)

- SHA originally designed by NIST & NSA in 1993
- Revised in 1995 as SHA-1 (NIST FIPS 180-1)
- Based on design of MD4
- Produces 160-bit hash values
- Recent 2005 analysis on security of SHA-1 have raised concerns on its use in future applications
- NIST issued revision FIPS 180-2 in 2002
  - Adds 3 additional versions of SHA (SHA-256, SHA-384, SHA-512)
  - Designed for compatibility with increased security provided by the AES cipher
  - Structure and detail is similar to SHA-1

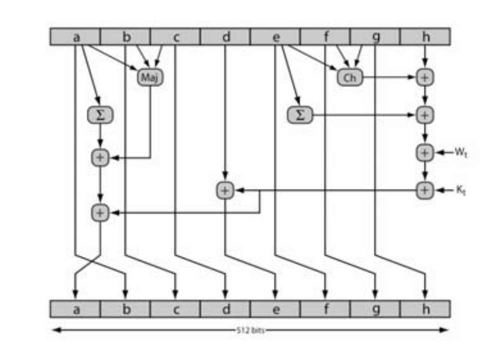
SHA-512 Overview



SHA-512 Compression Function

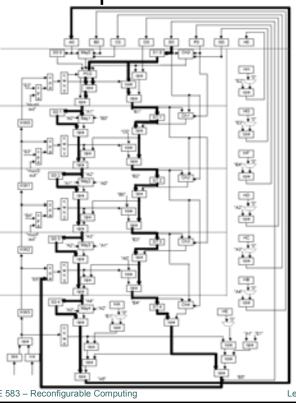
- Heart of the algorithm
- Processing message in 1024-bit blocks
- Consists of 80 rounds
  - Updating a 512-bit buffer
  - Using a 64-bit value *Wt* derived from the current message block
  - A round constant *Kt* that represents the first 64 bits of the fractional parts of the cube root of first 80 prime numbers

SHA-512 Round Function



1 Gbps SHA-512 Implementation

- Partial unrolling (5 rounds), pipelining
- 1 Gbps on Virtex-E FPGAs
- See [LieGre04A] for details

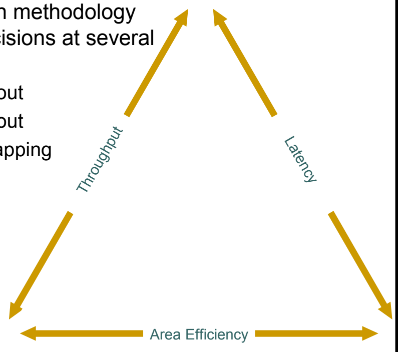


Application – Private-Key Crypto

- The Advanced Encryption Standard (AES) is becoming the block cipher of choice for private-key cryptography
- Implementing AES on FPGA hardware has been looked at in some depth:
  - Approximately 50 unique research implementations!
  - Various commercial cores (Actel, Helion Tech, Amphion, etc.)
- Approach taken – an exploration of the decisions that lead to area/delay tradeoffs in an AES FPGA implementation
- End result – pareto optimal designs in terms of throughput, latency, and area efficiency

General Approach [ZamNgu04A]

- Top-down design methodology incorporates decisions at several levels:
  - Inter-round layout
  - Intra-round layout
  - Technology mapping



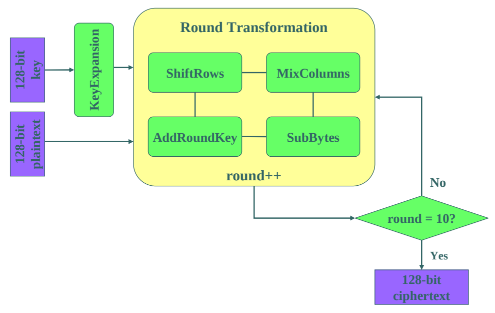
General Approach (cont.)

- General approach applied to an AES FPGA design targeting the Xilinx Virtex-II architecture
  - Familiarity with architecture and toolflow
  - All designs fit on Xilinx XC2V4000 or better
- Implemented using a single VHDL core with user directives driving the optimizations
- Results presented for AES-128E
  - Longer keys only require additional rounds
  - Decryption algorithm very similar to encryption

Overview of AES

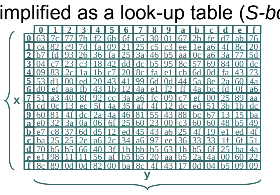
- In 1997 NIST announced an open competition for cipher designers to replace the aging Data Encryption Standard (DES)
  - 15 submissions
  - Publicly evaluated based on security, simplicity, and suitability for implementing in hardware and software
- Rijndael algorithm developed by Vincent Rijmen and Joan Daemen – selected as winner in 2000
- AES is Rijndael restricted to 128-bit blocks and keys of 128, 192, or 256 bits

AES-128E Algorithm



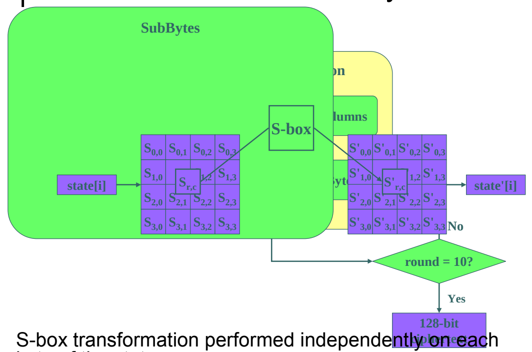
Overview of AES (cont.)

- 128-bit input is copied into a two-dimensional (4x4) byte array referred to as the *state*
  - Round transformations operate on the state array
  - Final state copied back into 128-bit output
- AES makes use of a non-linear substitution function that operates on a single byte
  - Can be simplified as a look-up table (*S-box*)



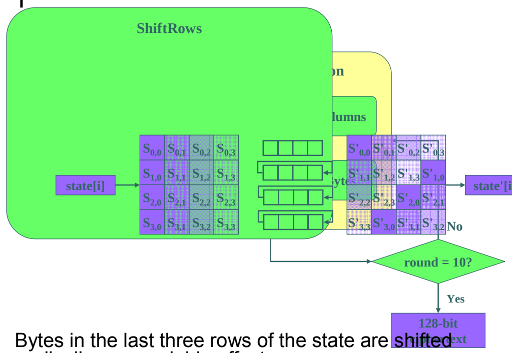
x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	09	21	25	c5	c3	ee	1e	a6	4f	8c	20
2	b7	fd	93	26	36	fa	25	3a	46	b5	aa	0c	a6	3a	77	5d
3	04	c7	23	c3	18	42	dd	dc	b5	95	8c	57	69	84	00	dc
4	09	83	2c	1a	1b	c7	20	8c	fa	e1	cb	6d	0d	fa	43	73
5	53	d1	00	ed	20	43	41	99	6d	0d	44	5a	8e	2a	60	4a
6	d0	ef	aa	fb	43	1b	12	4a	e1	f2	ff	4a	bc	fd	0f	a6
7	51	a3	40	8f	92	cc	3a	a6	fc	09	c7	ef	00	25	89	aa
8	cd	0c	13	ec	5f	4a	35	af	4f	12	dc	ed	51	3b	1b	dc
9	60	81	4f	dc	2a	4a	46	81	55	43	88	bc	67	13	15	ba
a	e0	32	3a	0a	06	6f	25	60	23	00	c3	60	60	48	b5	49
b	e7	c8	37	6d	d5	12	ed	45	43	a6	25	4f	19	e1	ed	4f
c	ba	25	25	2e	a6	2c	34	a6	97	ee	36	33	33	1f	6f	51
d	70	b5	b5	66	40	3f	1b	bb	b5	63	1b	b5	6f	25	ba	4a
e	e1	98	11	11	56	af	b5	b5	20	aa	b5	2a	4a	00	60	22
f	8c	89	0d	0d	82	00	ba	8c	4f	43	17	0d	04	b5	09	09

AES-128E Modules: SubBytes


SubBytes
S-box
state[i]
state'[i]
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{r,c}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S'_{0,0}$ $S'_{0,1}$ $S'_{0,2}$ $S'_{0,3}$
$S'_{1,0}$ $S'_{r,c}$ $S'_{1,2}$ $S'_{1,3}$
$S'_{2,0}$ $S'_{2,1}$ $S'_{2,2}$ $S'_{2,3}$
$S'_{3,0}$ $S'_{3,1}$ $S'_{3,2}$ $S'_{3,3}$
No
round = 10?
Yes
128-bit

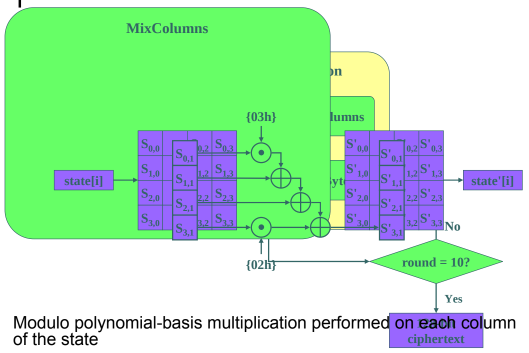
- S-box transformation performed independently on each byte of the state

AES-128E Modules: ShiftRows


ShiftRows
state[i]
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S'_{0,0}$ $S'_{0,1}$ $S'_{0,2}$ $S'_{0,3}$
$S'_{1,1}$ $S'_{1,2}$ $S'_{1,3}$ $S'_{1,0}$
$S'_{2,2}$ $S'_{2,3}$ $S'_{2,0}$ $S'_{2,1}$
$S'_{3,3}$ $S'_{3,0}$ $S'_{3,1}$ $S'_{3,2}$
state'[i]
No
round = 10?
Yes
128-bit

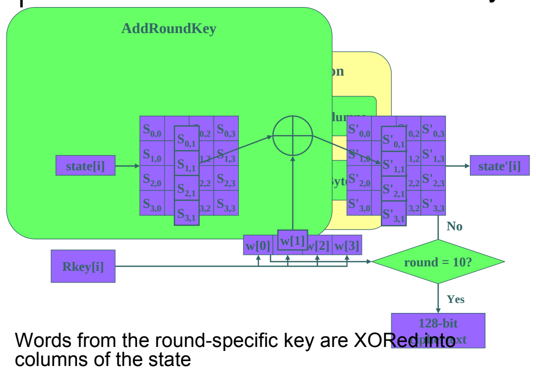
- Bytes in the last three rows of the state are shifted cyclically over variable offsets

AES-128E Modules: MixColumns


MixColumns
{03h}
{02h}
state[i]
$S_{0,0}$ $S_{1,0}$ $S_{2,0}$ $S_{3,0}$
$S_{0,1}$ $S_{1,1}$ $S_{2,1}$ $S_{3,1}$
$S'_{0,0}$ $S'_{1,0}$ $S'_{2,0}$ $S'_{3,0}$
$S'_{0,1}$ $S'_{1,1}$ $S'_{2,1}$ $S'_{3,1}$
state'[i]
No
round = 10?
Yes
ciphertext

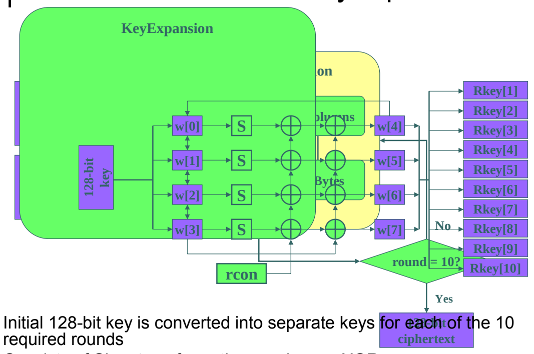
- Modulo polynomial-basis multiplication performed on each column of the state
- Can be simplified as series of AND and XOR operations

AES-128E Modules: AddRoundKey


AddRoundKey
state[i]
$S_{0,0}$ $S_{1,0}$ $S_{2,0}$ $S_{3,0}$
$S_{0,1}$ $S_{1,1}$ $S_{2,1}$ $S_{3,1}$
$S'_{0,0}$ $S'_{1,0}$ $S'_{2,0}$ $S'_{3,0}$
$S'_{0,1}$ $S'_{1,1}$ $S'_{2,1}$ $S'_{3,1}$
state'[i]
Rkey[i]
w[0] w[1] w[2] w[3]
No
round = 10?
Yes
128-bit

- Words from the round-specific key are XORed into columns of the state

AES-128E Modules: KeyExpansion



- Initial 128-bit key is converted into separate keys for each of the 10 required rounds
- Consists of Sbox transformations and some XORs

Design Decisions

- Online/offline key generation
- Inter-round layout decisions
  - Round unrolling
  - Round pipelining
- Intra-round layout decisions
  - Transformation pipelining
  - Transformation partitioning
- Technology mapping decisions
  - S-box synthesis as Block SelectRAM, distributed ROM primitives, or logic gates

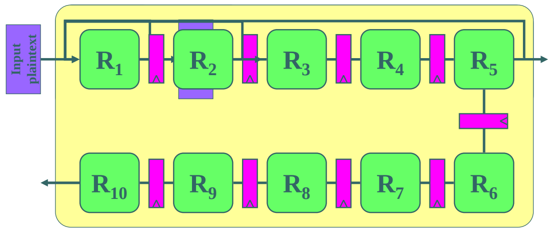
Round Unrolling / Pipelining

- *Unrolling* replaces a loop body (round) with *N* copies of that loop body
- AES-128E algorithm is a loop that iterates 10 times – N є [1, 10]
  - *N* = 1 corresponds to original looping case
  - *N* = 10 is a fully unrolled implementation
- *Pipelining* is a technique that increases the number of blocks of data that can be processed concurrently
  - Pipelining in hardware can be implemented by inserting registers
  - Unrolled rounds can be split into a certain number of pipeline stages
- These transformations will increase throughput but increase area and latency

Round Unrolling / Pipelining (cont.)

Unrolling factor = 10

Round pipelining = ON



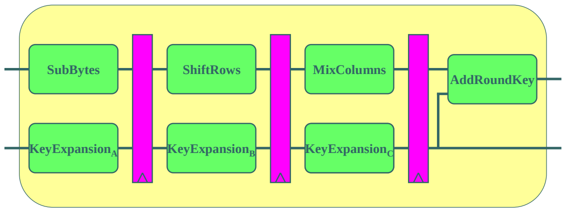
Transformation Partitioning

- FPGA maximum clock frequency depends on critical logic path
  - Inter-round transformations can't improve critical path
  - Individual transformations can be pipelined with registers similar to the rounds
- Transformations that are part of the maximum delay path can be partitioned and pipelined as well
- Can result in large gains in throughput with only minimal area increases

Partitioning / Pipelining (cont.)

Transformation pipelining = ON

Transformation partitioning = ON



S-box Technology Mapping

- With synthesis primitives, can map the S-box lookup tables to different hardware components
- Two S-boxes can fit on a single Block SelectRAM

```
constant SSYNROMSTYLE: string := “select_rom”; -- {logic, select_rom}
entity Sbox is

  port(BYTE_IN  : in std_logic_vector(7 downto 0);
       BYTE_OUT : out std_logic_vector(7 downto 0));

  attribute syn_romstyle : string;
  attribute syn_romstyle of BYTE_OUT : signal is SSYNROMSTYLE;

end Sbox;
...
```

Sample VHDL code

Experimental Setup

- FPGA target – Xilinx XC2V4000
  - Medium-sized member of the Virtex-II device family
  - 5760 CLBs (equivalent to 23040 slices)
  - 120 Block SelectRAM modules, each can hold up to 18 Kbits of data
- Synplify Pro 7.2.1 from Synplicity used for synthesis
- ISE 5.2i from Xilinx used for the place-and-route and timing analysis

Experimental Setup (cont.)

- For each design we measured:
  - Maximum possible clock rate – f_{clk}
  - Number of utilized slices – N_{slice}
  - Number of utilized SelectRAMs – N_{bram}
- From these base statistics we calculated maximum throughput (*Tput*) and the latency to encrypt a single block (*Lat*)
- Some idea about the area efficiency can be obtained by analyzing the following metric:

$$Eff = Tput \,/\, N_{slice} \,,$$

measured in throughput rate (bps) per slice

Area and Performance Results

- Each design is labeled UF*X*-PP*YZ*:
  - *X* – unrolling factor, *X* ϵ {1, 2, 5, 10}
  - *Y* – amount of transformation partitioning and pipelining:
    - For *Y* = 0 the design has no pipelining
    - For *Y* = 1 each unrolled round is pipelined
    - For *Y* = 2 each round is split into two stages
    - For *Y* = 3 each round is split into three stages
  - *Z* – the S-box technology mapping
    - *Z* = [B] uses Block SelectRAMs
    - *Z* = [D] uses distributed ROM primitives
    - *Z* = [L] instantiates logic gates

Results: Observed Trends

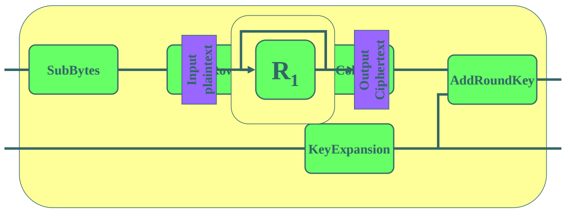
Design	f_{clk} (MHz)	N_{slice}	N_{bram}	*Tput* (Gbps)	*Lat* (ns)	*Eff* ($\frac{\text{Mbps}}{\text{slice}}$)
UF10-PP3D	**184.16**	16938	0	**23.57**	162.9	1.39

Design	f_{clk} (MHz)	N_{slice}	N_{bram}	*Tput* (Gbps)	*Lat* (ns)	*Eff* ($\frac{\text{Mbps}}{\text{slice}}$)
UF1-PP0B	110.16	**387**	10	1.41	90.78	3.64
UF1-PP0D	77.91	1780	0	1.00	128.4	0.56
UF1-PP0L	59.00	2744	[illegible]	[illegible]	[illegible]	[illegible]
UF1-PP3D	178.09	1940	0	2.28	168.5	1.18
UF1-PP3L	147.75	2909	0	1.89	203.0	0.65
UF2-PP1B	118.57	753	20	3.04	84.34	4.04
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
UF2-PP2D	119.96	3445	0	3.07	166.7	0.89
[illegible]	[illegible]	[illegible]	20	4.44	173.0	3.54
UF2-PP3L	118.30	5570	0	3.03	253.6	0.54
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
UF5-PP1D	68.521	7995	0	4.39	145.9	0.55
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
UF5-PP2L	76.26	11974	0	4.88	262.3	0.41
[illegible]	[illegible]	2810	[illegible]	11.12	172.7	3.96
UF10-PP1B	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
UF10-PP1D	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	22.93	111.6	**6.09**
UF10-PP3B	183.58	4901	100	23.50	163.4	4.79
UF10-PP3D	**184.16**	16938	0	**23.57**	162.9	1.39

- Unrolling increases the number of slices by a significant amount
- For the S-boxes, Block SelectRAMs perform slightly worse than the distributed ROM primitives, but there is a considerable savings in slice usage
- Aggressive transformation partitioning is effective in increasing throughput

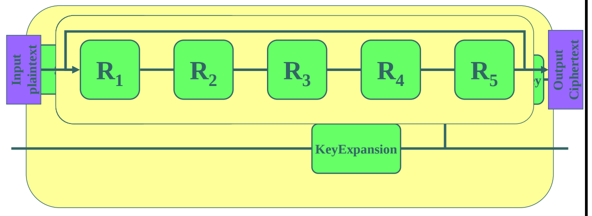
Results: UF1-PP0B

Design	f_{clk} (MHz)	N_{slice}	N_{bram}	*Tput* (Gbps)	*Lat* (ns)	*Eff* ($\frac{\text{Mbps}}{\text{slice}}$)
UF1-PP0B	110.16	**387**	10	1.41	90.78	3.64



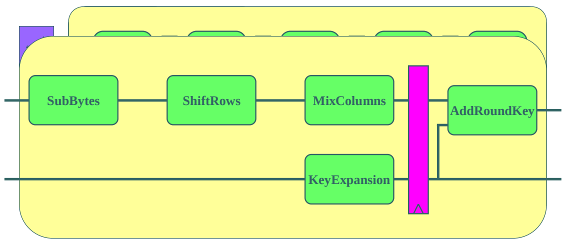
Results: UF5-PP0B

Design	f_{clk} (MHz)	N_{slice}	N_{bram}	*Tput* (Gbps)	*Lat* (ns)	*Eff* ($\frac{\text{Mbps}}{\text{slice}}$)
UF5-PP0B	72.438	1532	50	4.64	**27.61**	3.03



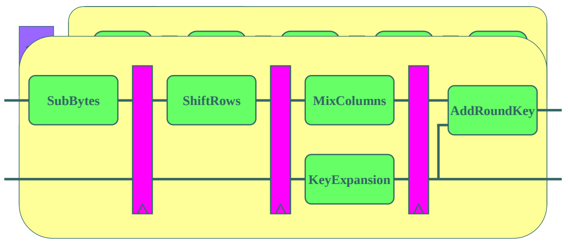
Results: UF10-PP2B

Design	f_{clk} (MHz)	N_{slice}	N_{bram}	*Tput* (Gbps)	*Lat* (ns)	*Eff* ($\frac{\text{Mbps}}{\text{slice}}$)
UF10-PP2B	179.147	3766	100	22.93	111.6	**6.09**



Results: UF10-PP3D

Design	f_{clk} (MHz)	N_{slice}	N_{bram}	*Tput* (Gbps)	*Lat* (ns)	*Eff* ($\frac{\text{Mbps}}{\text{slice}}$)
UF10-PP3D	**184.16**	16938	0	**23.57**	162.9	1.39

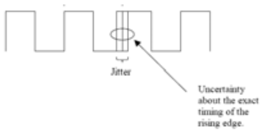


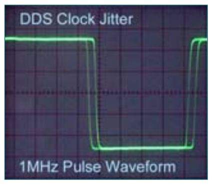
Application – Random Number Generation

- Cryptographic applications often require good sources of random numbers:
  - Key generation
  - Initialization vectors
- Types of random number generators:
  - Pseudo-Random Number Generators (PRNG) – appear to be random, initialized with an externally generated sequence (deterministic)
  - Cryptographically Secure PRNGs (CSPRNG) – a PRNG where prediction of the next input bit given a previously-generated sequence is computationally intractable
  - True Random Number Generators (TRNG) – output is based on some underlying physical random process

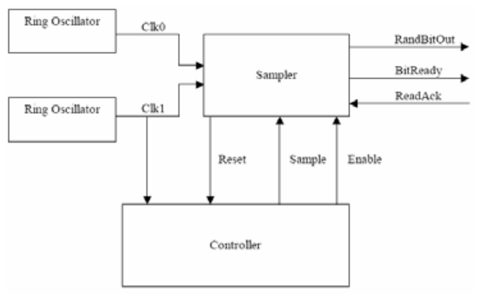
The Method [KohGaj04A]

- Make use of the *clock jitter* in a circuit:
  - Variation of the significant instants of the clock
  - Nondeterministic, may have many sources:
    - Semiconductor noise
    - Crosstalk
    - Power supply variations
    - Electro-magnetic fields

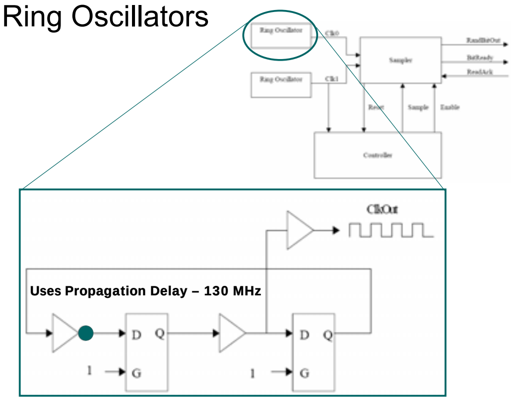




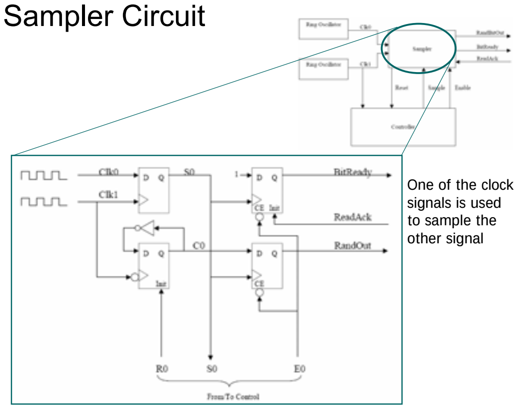
Overall Design



Ring Oscillators



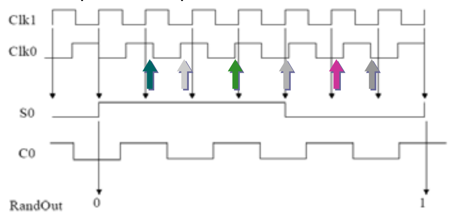
Sampler Circuit



One of the clock signals is used to sample the other signal

Sampler Output

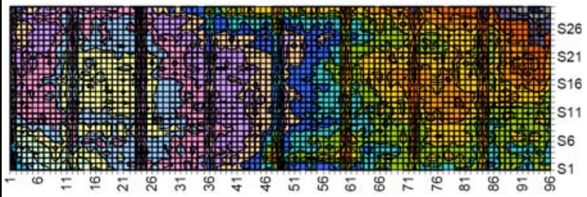
- Clock Skew (jitter) in between two clock signals is used (e.g. sampled) to generate a totally random bit
- The output clock skew:
  - Will never be uniform
  - Is not simple out-out-phase behavior



Good Speed Ratios

- Ring oscillators with closely matched frequencies require that a desired speed ratio must be achieved
- What factors affect this achievement?
  - Variation in CLB speed
    - 7% difference between the slowest CLB and the fastest one
    - Sensitive to temperature and difficult for measurement
  - Variation in the frequency of an oscillator with the chip temperature
    - Close placement
    - To use a large number of oscillators

CLB Speed / Temperature Variation



Summary

- FPGA platforms are a popular choice for implementing cryptographic applications
  - High throughputs
  - Relatively low design cost
  - Algorithmic agility / upload
- Many other algorithms have been implemented that we haven't discussed today:
  - Public-key cryptography (e.g. RSA, ECC)
  - Private-key cryptography (e.g. DES, 3DES)
  - Cryptographic hash functions (e.g. MD5, RIPEMD)
- Security issues as they pertain to using FPGAs have not been fully addressed