VLSI design

Just for fun: LFSRs, project ideas

P.J. Drongowski
SandSoftwareSound.net

Unbreakable encryption scheme

- Basic scheme
  - Given the text of a message to be sent (clear text)
  - Given an infinite sequence of random numbers
  - Sender and receiver have a copy of the sequence
  - The sequence is the message "key"
  - Generate encrypted text of the message
  - "Combine" each character in clear text with its corresponding key value in the random sequence
  - Send the message
  - Receiver "uncombines" using the same key values
  - Key values are discarded and never used again

- Related notions
  - "One time pad"
  - Teletype communication scheme due to _____
    - Key values are recorded on infinite tape
    - Sender and receiver have the same tape
    - Values and characters in binary representation
    - "Combine" and "uncombine" is exclusive OR

- Problem: Generate infinite stream of random numbers

- Pseudo-random numbers (PRN)
  - Uniformly distributed
  - Periodic (long period is *very* desirable)

- PRN generation
  - Modulus - residue
  - Linear feedback shift register (LFSR)

Simple encryption program

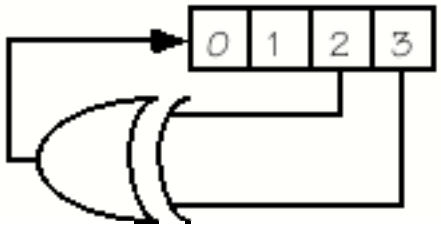
```
#define NKEYS 16

unsigned char StreamOfKeys[NKEYS] =

  {
  0x73, 0x12, 0x34, 0x09, 0x64, 0x32, 0x39, 0x2C,
  0x0E, 0x33, 0x1B, 0x2F, 0x03, 0x1A, 0x37, 0x15
  } ;

char TestString[]
  = "The quick brown fox jumped over the lazy dog.'

int NextInStream = 0 ;

void AdvanceToNextKey()
  {
  NextInStream++ ;
  if (NextInStream >= NKEYS) NextInStream = 0 ;
  }

char Encrypt(symbol) char symbol ;
  {
  return( symbol ^ StreamOfKeys[NextInStream] ) ;
  }

char Decrypt(symbol) char symbol ;
  {
  return( symbol ^ StreamOfKeys[NextInStream] ) ;
  }

main()
  {
  register char *c ;

  for (c = TestString ; *c ; c++)
    {
    printf("%c\t%d\t%c\n", *c,
      Encrypt(*c), Decrypt(Encrypt(*c))) ;
    AdvanceToNextKey() ;
    }
  }
```

Sample run (simple program)

```
T 39 T
h 122 h
e 81 e
  41
q 21 q
u 71 u
i 80 i
c 79 c
k 101 k
  19
b 121 b
r 93 r
o 108 o
w 109 w
n 89 n
  53
f 21 f
o 125 o
x 76 x
  41
j 14 j
u 71 u
m 84 m
p 92 p
e 107 e
d 87 d
  59
o 64 o
v 117 v
e 127 e
r 69 r
  53
t 7  t
h 122 h
e 81 e
  41
l 8  l
a 83 a
z 67 z
y 85 y
  46
d 87 d
o 116 o
g 72 g
. 45 .
```

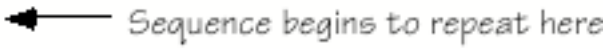
4-bit LFSR program

```
/*
 * Four bit LFSR
 * Feed bits 3 and 2 back into input
 */

int LFSR = 1 ; /* Never ever zero! */

#define Bit3(A) ((A >> 3) & 1)
#define Bit2(A) ((A >> 2) & 1)

void ComputeNextValue()

  {
  LFSR =
    ((LFSR << 1) | (Bit3(LFSR) ^ Bit2(LFSR))) & 0x0F
  }

main()

  {
  int i ;
  for (i = 0 ; i < 32 ; i++)
    {
    printf("Step: %d Key: %d\n", i, LFSR) ;
    ComputeNextValue() ;
    }
  }
```



Sample run (4-bit LFSR)

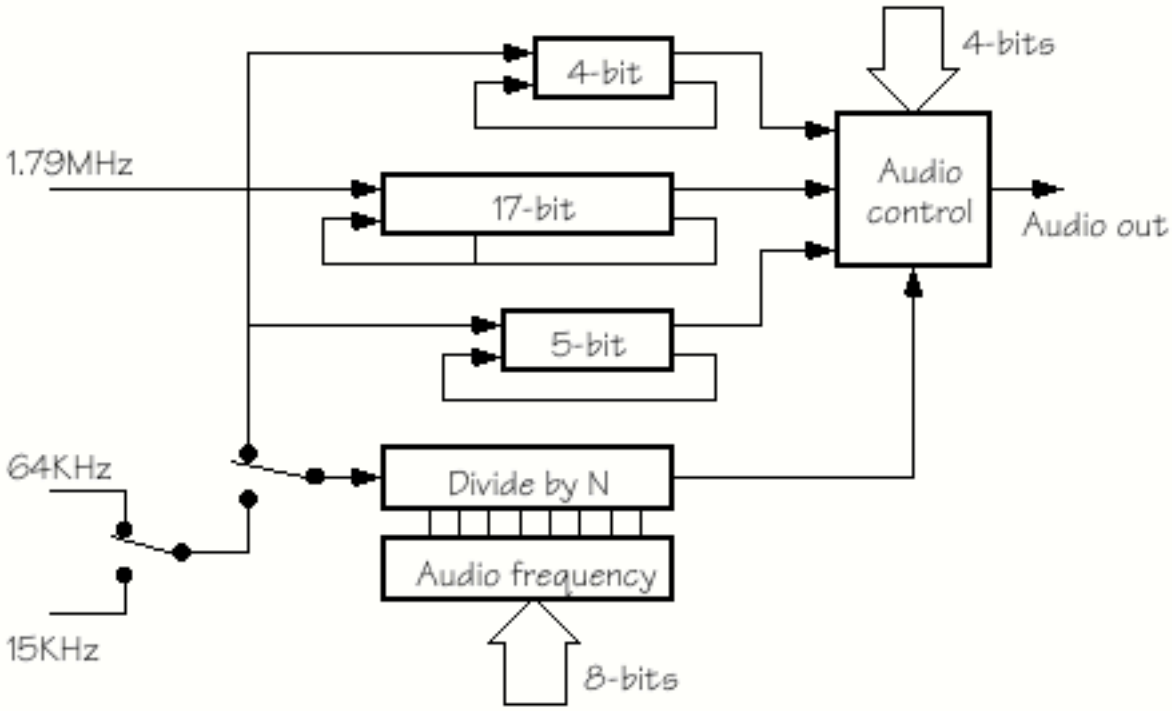
```
Step:  0 Key:  1
Step:  1 Key:  2
Step:  2 Key:  4
Step:  3 Key:  9
Step:  4 Key:  3
Step:  5 Key:  6
Step:  6 Key: 13
Step:  7 Key: 10
Step:  8 Key:  5
Step:  9 Key: 11
Step: 10 Key:  7
Step: 11 Key: 15
Step: 12 Key: 14
Step: 13 Key: 12
Step: 14 Key:  8
Step: 15 Key:  1
Step: 16 Key:  2
Step: 17 Key:  4
Step: 18 Key:  9
Step: 19 Key:  3
Step: 20 Key:  6
Step: 21 Key: 13
Step: 22 Key: 10
Step: 23 Key:  5
Step: 24 Key: 11
Step: 25 Key:  7
Step: 26 Key: 15
Step: 27 Key: 14
Step: 28 Key: 12
Step: 29 Key:  8
Step: 30 Key:  1
Step: 31 Key:  2
```



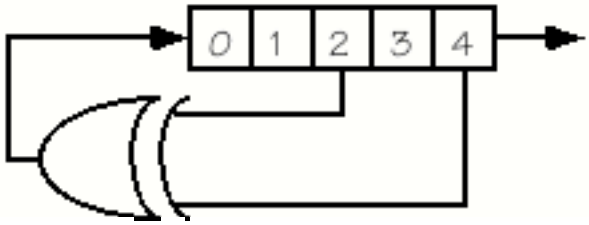
Sequence is 15 values long

Atari 800 sound generation

- Four audio channels
- Channels can be chained to increase resolution
- "Divide by N" counters act as frequency generators
- LFSR's (polynomials) provide noise sources



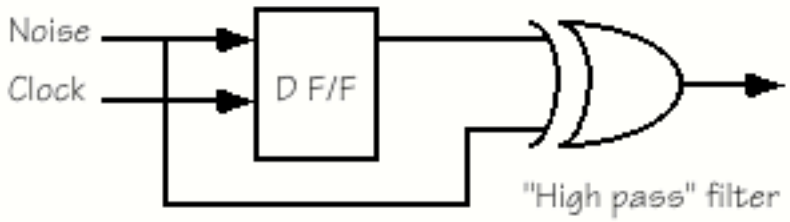
- 5-bit LFSR
- Short patterns - drones, motors, repetitive sounds
- Long patterns - explosions, steam, white noise



Atari 800 "filtering"

- "Low pass" filter
  - Pass frequencies below cutoff (corner) frequency
  - Sample LFSR output under control of freq divider
  - Acts as low pass frequency clock
  - Divide by N counter sets maximum noise frequency
  - Just need to AND signals together

- "High pass" filter
  - Pass frequencies above cutoff frequency
  - Use D-type flip/flop and XOR gate
  - If input changes faster than clock
    → Output will tend to follow input
  - If input is slower than clock
    → Output will not pass very often
  - Forms crude high pass filter
  - Minimum frequency is set by clock rate



- Control bits
  - Audio frequency (8-bits)
  - Enable/disable high pass filter
  - 15KHz / 64KHz clock
  - 1.79KHz / slow clock select
  - 17-bit / 9-bit polynomial