AES Standard Notes

Section 3

input bytes					*state array*					*output bytes*			
in_0	in_4	in_8	in_{12}	→	$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	→	out_0	out_4	out_8	out_{12}
in_1	in_5	in_9	in_{13}		$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$		out_1	out_5	out_9	out_{13}
in_2	in_6	in_{10}	in_{14}		$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$		out_2	out_6	out_{10}	out_{14}
in_3	in_7	in_{11}	in_{15}		$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$		out_3	out_7	out_{11}	out_{15}

Figure 1. State array input and output

128 bit inputs → 16 bytes, labelled in bytes $a_0 \dots a_{15}$ and bits $a_i = r_0 \dots r_7$ in form → 4 words, V_0 V_1 V_2 V_3 words $V[0..3]$

Section 4:

$10110011 \rightarrow x^7+x^5+x^4+x+1$

addition) is XOR style, so $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$

equv. to subtraction

multiplication) $b \cdot c = b(x) \cdot c(x) \mod m(x)$ for $M(x) = x^7+x^4+x^3+x+1$ ← hex

consequently, it is useful to consider the special case that $c(x) = x$ (i.e., $c = \{02\}$). In particular, the product $b \bullet \{02\}$ can be expressed as a function of b, denoted by $\text{XTIMES}(b)$, as follows:

$$\text{XTIMES}(b) = \begin{cases} \{b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0\, 0\} & \text{if } b_7 = 0 \\ \{b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0\, 0\} \oplus \{0\,0\,0\,1\,1\,0\,1\,1\} & \text{if } b_7 = 1. \end{cases} \quad (4.5)$$

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

multiplicative inverses: let $b \cdot b^{-1} = \{01\}$, or just 1.

$b^{-1} = b^{254}$

Section 5:

Algorithm 1 Pseudocode for CIPHER()

```
1:  procedure CIPHER(in, Nr, w)
2:      state ← in                                   ▷ See Sec. 3.4
3:      state ← ADDROUNDKEY(state, w[0..3])          ▷ See Sec. 5.1.4
4:      for round from 1 to Nr − 1 do
5:          state ← SUBBYTES(state)                  ▷ See Sec. 5.1.1
6:          state ← SHIFTROWS(state)                 ▷ See Sec. 5.1.2
7:          state ← MIXCOLUMNS(state)                ▷ See Sec. 5.1.3
8:          state ← ADDROUNDKEY(state, w[4 ∗ round..4 ∗ round + 3])
9:      end for
10:     state ← SUBBYTES(state)
11:     state ← SHIFTROWS(state)
12:     state ← ADDROUNDKEY(state, w[4 ∗ Nr..4 ∗ Nr + 3])
13:     return state                                 ▷ See Sec. 3.4
14: end procedure
```

5.1.1) SubBytes() output byte $b' = \text{Sbox}(b)$

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \hat{b}_0 \\ \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \\ \hat{b}_4 \\ \hat{b}_5 \\ \hat{b}_6 \\ \hat{b}_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

where $\hat{b}_x = \begin{cases} \{00\} & \text{if } b = \{00\} \\ b^{-1} & \text{else} \end{cases}$

or, can use lookup table! for $b = \{xy\}$, $b' =$:

ex if $b = 53$, $b' = ed$

Table 4. SBOX(): substitution values for the byte xy (in hexadecimal format)

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	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

5.1.2) ShiftRows()

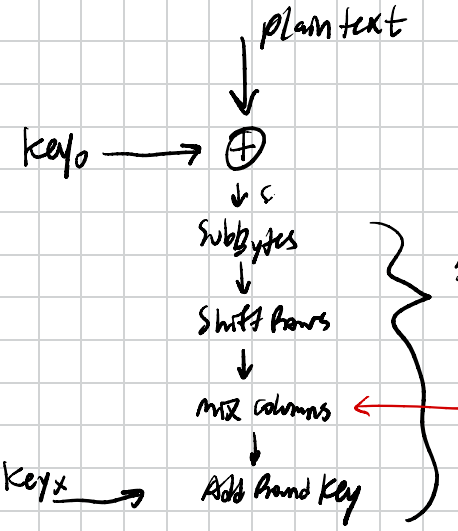
AES Video: 128 bit message —encrypt→ 128 bit output message

↑ uses a key of length

AES 128, AES 256, AES 192

for AES x, x bit key

*AES is symmetric! same key to encrypt and decrypt



128 : 10 Rounds

192 : 12 rounds

256 : 14 Rounds

Galois Field Idea:

for field: 0000-0000

↓

1111-1111

all operations $(+, -, \times, \div, b^{-1})$ yield another unit in the field. no overflow, etc.

key gets expanded by keyschedule into key$_0$ pieces

mix row mixes each row in a set way, internally. same elements in each row, just diff. order

mix columns is funky galois field multiplications/additions etc. which actually flips/scrambles bits

Lab 7 overview:

- could implement these things as big comb blocks... but too big!
- Instead, we will use FPGA Embedded RAM blocks to make it sequential

plaintext → (encipher) → ciphertext → (decipher) → plaintext

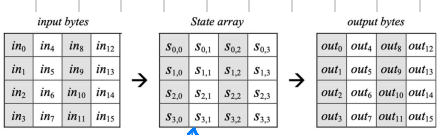
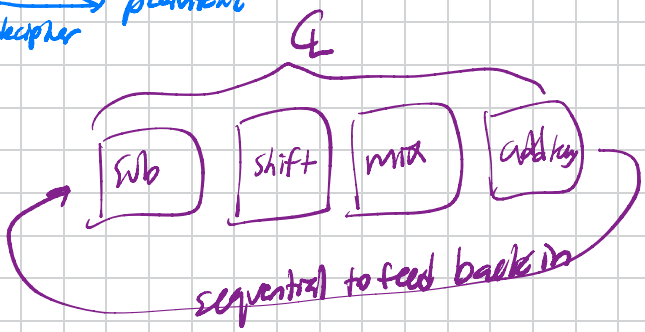


Figure 3: State array input and output.

math happens using these

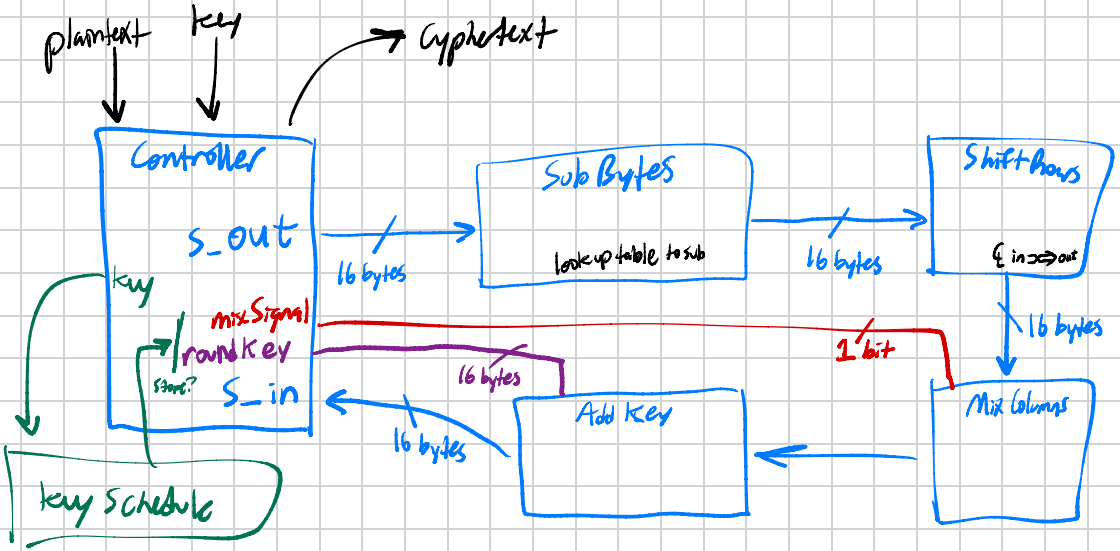


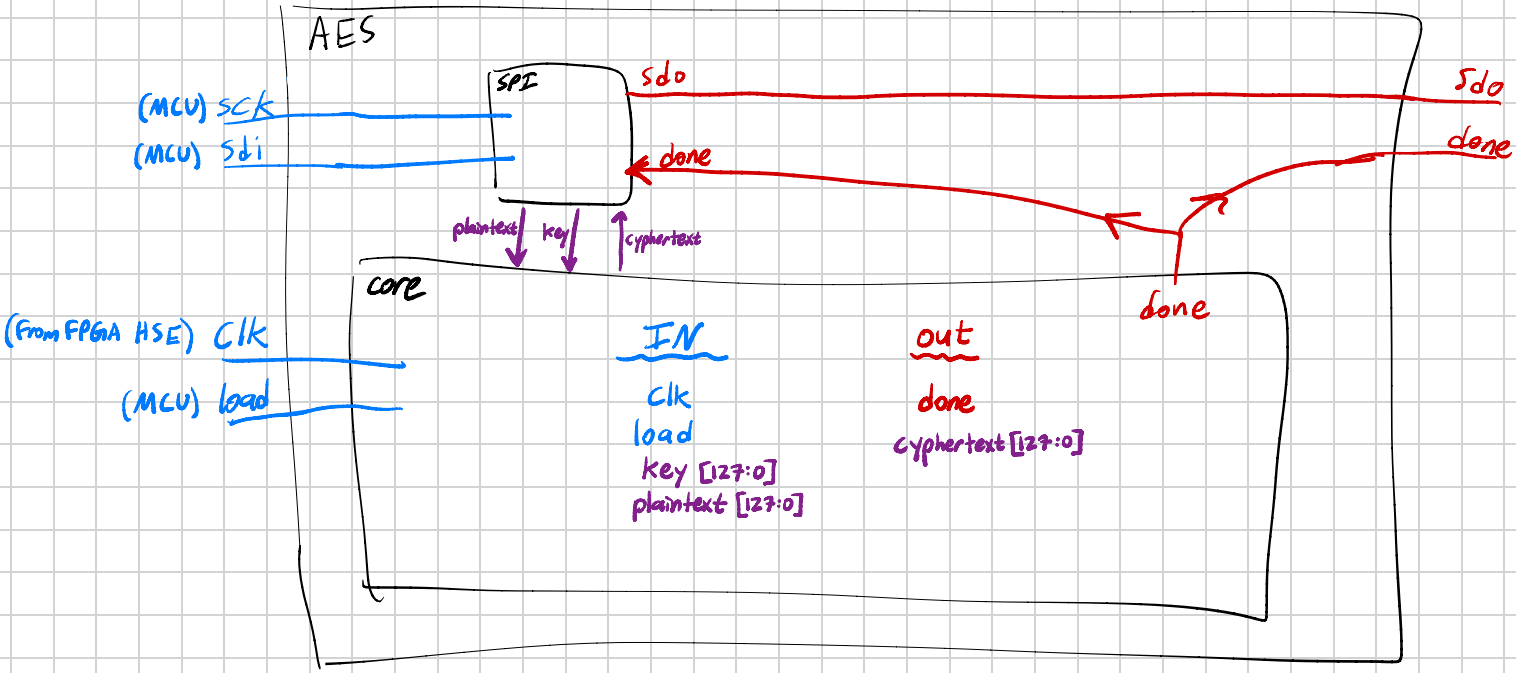
key expansion

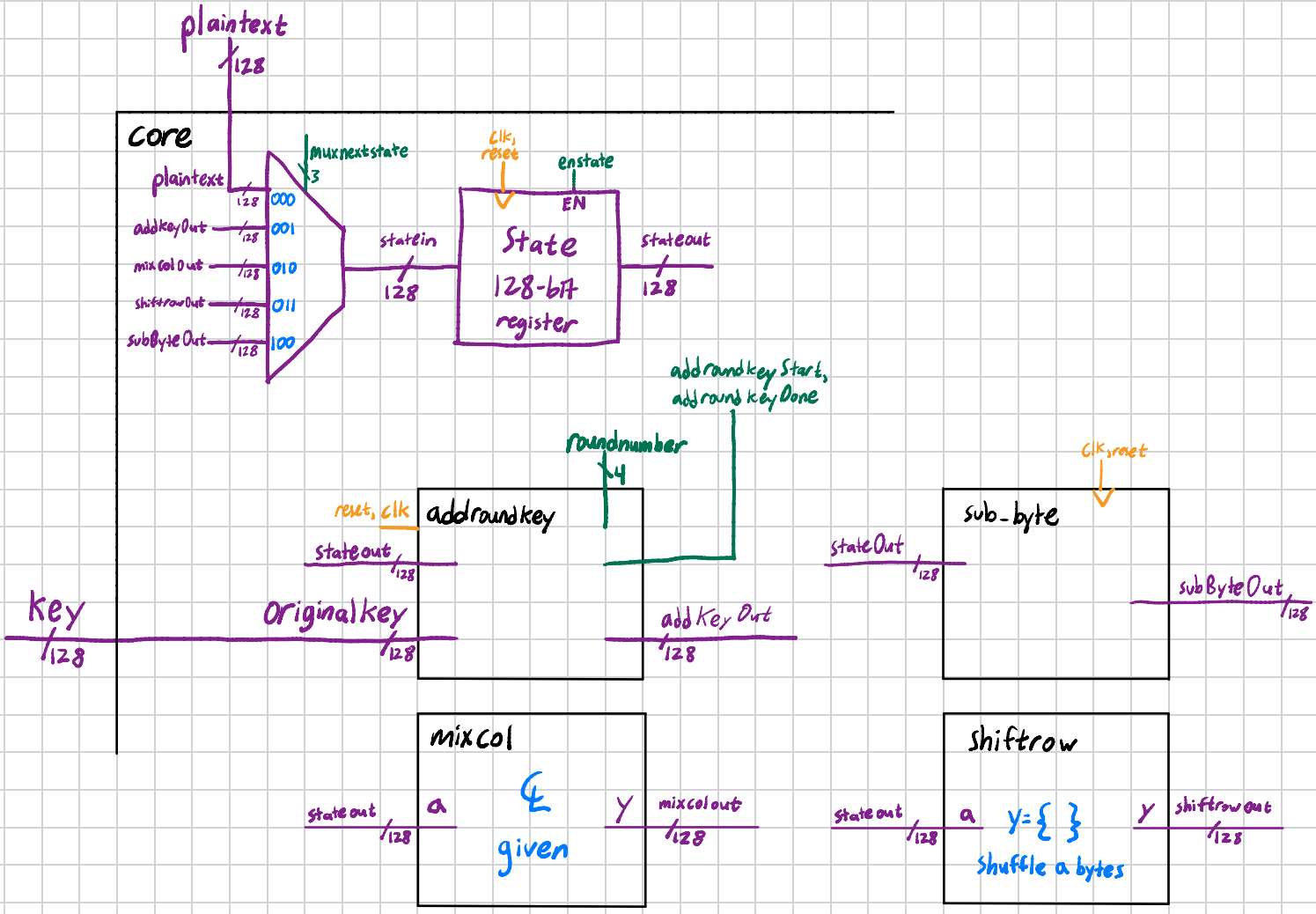
```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

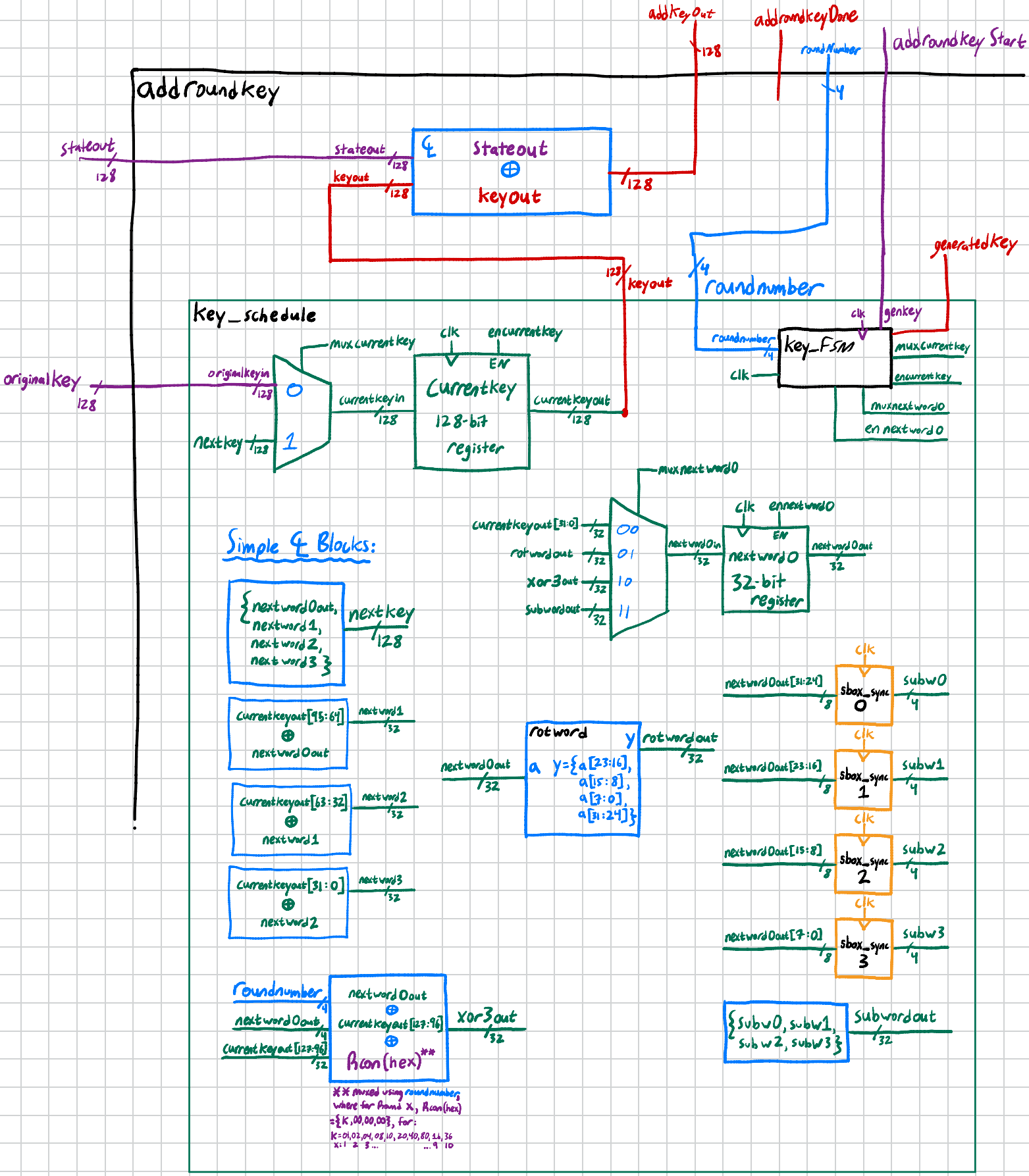
- we will also do key expansion!

Stab at a Block Diagram:




AES
SPI
(MCU) sck
(MCU) sdi
sdo
done
Sdo
done
plaintext
key
cyphertext
core
done
(From FPGA HSE) Clk
(MCU) load
IN
Clk
load
key [127:0]
plaintext [127:0]
out
done
cyphertext [127:0]


plaintext
128
core
plaintext
128
addkeyOut
128
mixcolOut
128
shiftrowOut
128
subByteOut
128
muxnextstate
3
000
001
010
011
100
statein
128
clk, reset
enstate
EN
State
128-bit
register
stateout
128
addroundkeyStart,
addroundkeyDone
roundnumber
4
reset, clk
addroundkey
stateout
128
key
128
Originalkey
128
addKeyOut
128
clk, reset
sub-byte
stateOut
128
subByteOut
128
mixcol
stateout
128
a
y
mixcolout
128
given
Shiftrow
stateout
128
a
y={ }
y
shiftrowout
128
Shuffle a bytes


addKeyOut
addroundkeyDone
roundNumber
addroundkey Start
addroundkey
stateout
128
stateout
128
keyout
128
4
Stateout
⊕
keyout
128
generatedkey
128
keyout
4
roundnumber
key_schedule
clk
genkey
roundnumber
4
key_FSM
clk
muxcurrentkey
encurrentkey
muxnextword0
ennextword0
originalkey
128
originalkeyin
128
muxcurrentkey
clk
encurrentkey
EN
Currentkey
128-bit
register
0
1
nextkey
128
currentkeyin
128
currentkeyout
128
muxnextword0
clk
ennextword0
currentkeyout[31:0]
32
rotwordout
32
xor3out
32
subwordout
32
00
01
10
11
nextword0in
32
EN
nextword0
32-bit
register
nextword0out
32
Simple 4 Blocks:
{nextword0out, nextword1, nextword2, nextword3}
nextkey
128
currentkeyout[95:64]
⊕
nextword0out
nextword1
32
currentkeyout[63:32]
⊕
nextword1
nextword2
32
currentkeyout[31:0]
⊕
nextword2
nextword3
32
rotword
a
y
y={a[23:16], a[15:8], a[7:0], a[31:24]}
nextword0out
32
rotwordout
32
clk
nextword0out[31:24]
8
sbox_sync
0
subw0
4
clk
nextword0out[23:16]
8
sbox_sync
1
subw1
4
clk
nextword0out[15:8]
8
sbox_sync
2
subw2
4
clk
nextword0out[7:0]
8
sbox_sync
3
subw3
4
{subw0, subw1, subw2, subw3}
subwordout
32
roundnumber
4
nextword0out
4
currentkeyout[127:96]
32
nextword0out
⊕
currentkeyout[127:96]
⊕
Rcon(hex)**
xor3out
32
** mixed using roundnumber, where for Round X, Rcon(hex) ={K,00,00,00}, for:
K=01,02,04,08,10,20,40,80,1b,36
X: 1 2 3 ... ... 9 10

Steps

0:
AddRoundKey

1-9:
SubBytes
ShiftRows
MixColumns
AddRoundKey

10:
SubBytes
ShiftRows
–
AddRoundKey

ShiftRows:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

→

0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

Key Schedule:

Repeat

f0	a	b	c	f1	n1	n2	n3	f2	n4	n5	n6
0	4	8	12	sp0				sp4			
1	5	9	13	sp1				sp5			
2	6	10	14	sp2				sp6			
3	7	11	15	sp3				sp7			

1) RotWord

13
14
15
12

2) SubBytes

13
14
15
12

3) XOR

0	13	X	
1	14	00	=
2	15	00	
3	12	00	
f0		Rcon	

$n1 = a \oplus sp0\text{-}3$

$n2 = b \oplus n1$

$n3 = c \oplus n2$

rot
sub
$f1 \oplus x \oplus rcon$
next x

x = 01, 02, 04, 08, 10, 20, 40, 80, 1b, 36

	actual	desired
B0	D9	39
	1101 1001	0011 1001
B1	BD	25
	1011 1101	0010 0101