ECE 385 – Digital Systems Laboratory

Lecture 17 –More Experiment 9

Zuofu Cheng

Fall 2018

Link to Course Website

ECE ILLINOIS

ILLINOIS

Experiment 9 Week 1 Demo Points

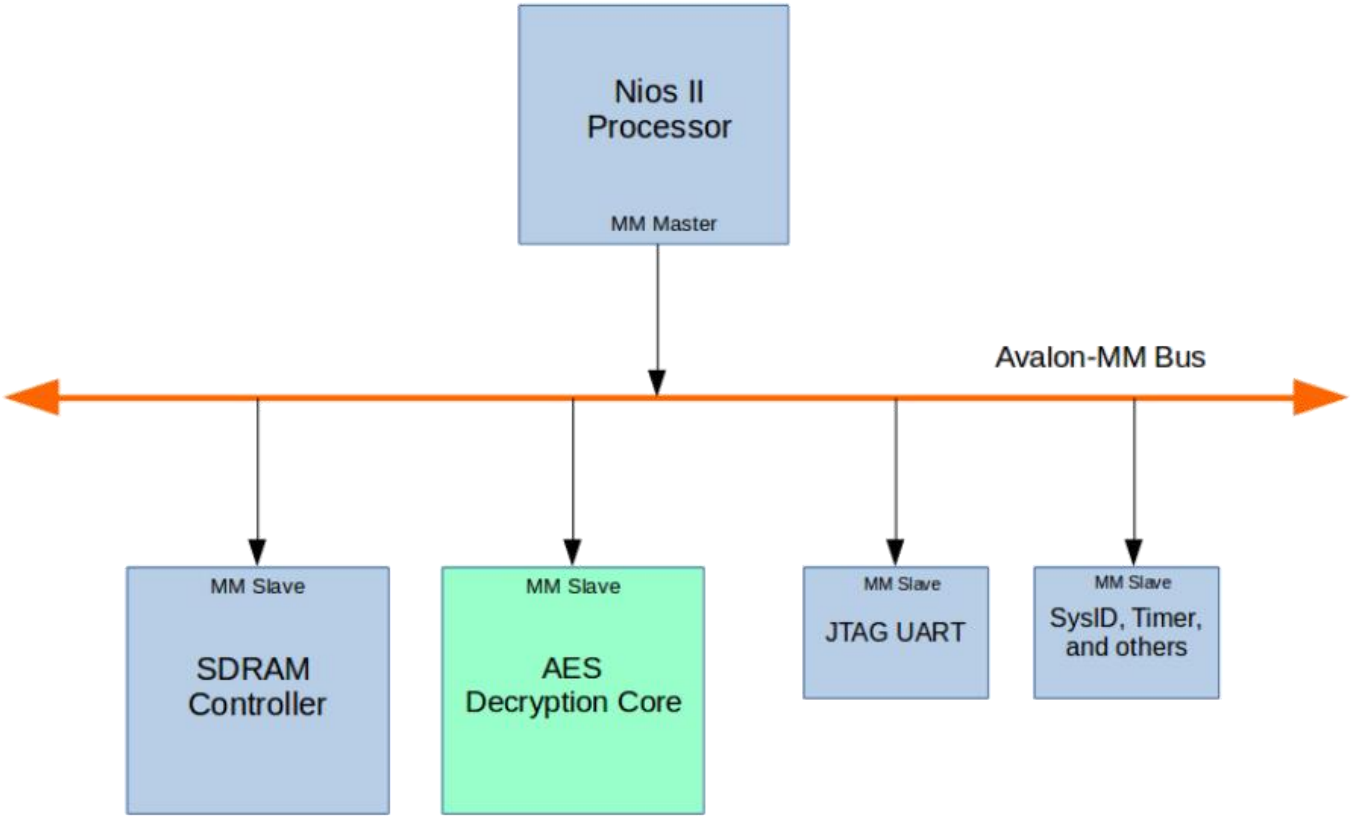
- 1 point: Correct key expansion
- 1 point: Correct intial round key and looping rounds operations
- 1 point: Correct AES encryption test vectors
- 1 point: Benchmark of NIOS II/e based AES in kB/s
- 1 point: Correct Hardware Software communication to display the correct key on the hex displays

Experiment 9 Week 2 Demo Points

- 1 point: Correctness of the looping rounds operations.
- 1 point: Correctness of stepping through the looping rounds using state machine. Specifically, there can only be one InvMixColumns instantiation.
- 1 point: Correctness of transmitting correct result to CPU and displaying plaintext via terminal.
- 1 point: Able to run consecutive decryption operations correctly without restarting the program.
- 1 point: Benchmark result for HDL based AES decryption in kB/s. You will need to explain what experiment(s) you ran to get this result.

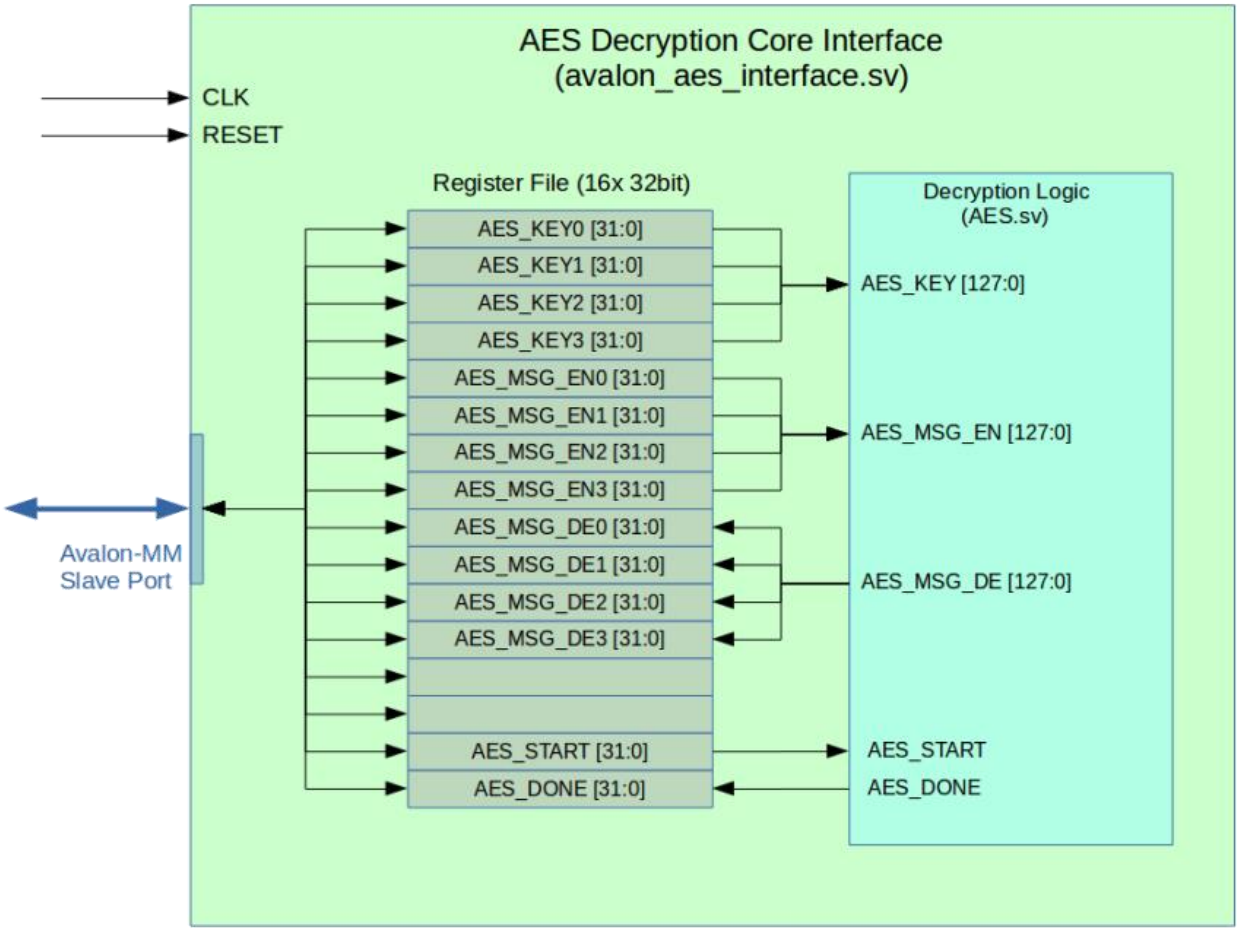
Nios II System Components in Lab 9

- Create a Avalon MM interface
- Decode 16 registers to use for HW/SW communication



AES Decryption Core Interface

- 16 Registers, each 32-bits (you need to write this)
- How many bytes on Avalon bus?



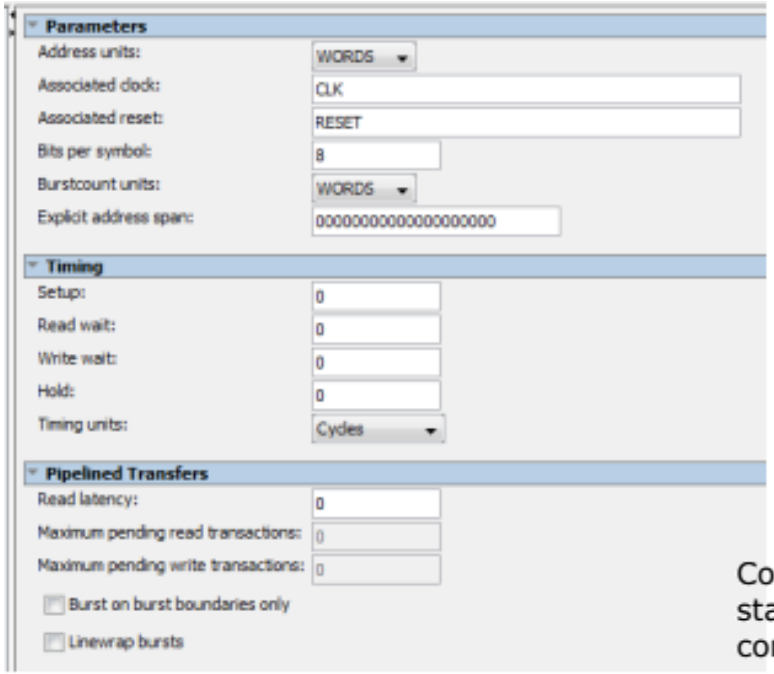
Avalon MM Interface Signals

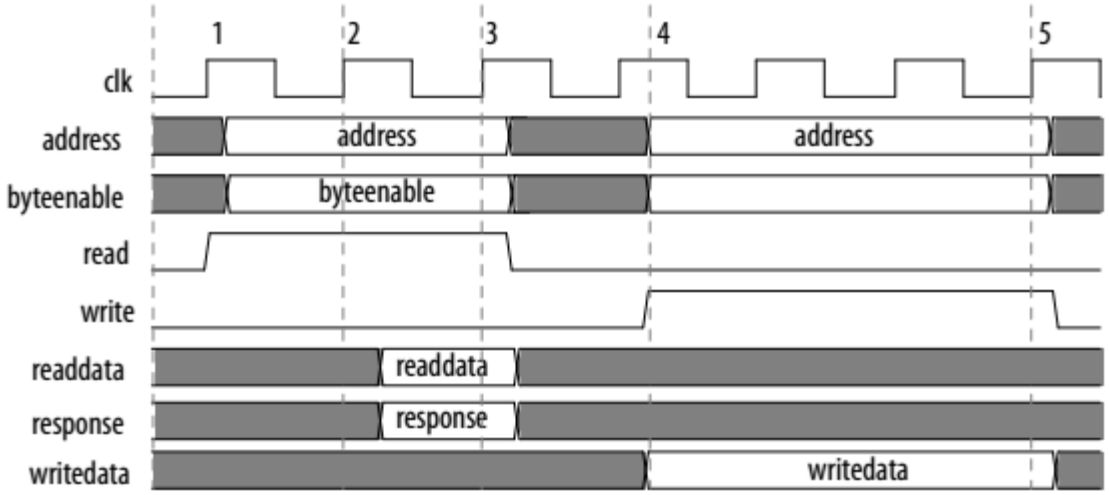
- You will create a module which will decode Avalon bus (following chart) and place data into 16 registers
- Note address is only 4 bits, don't need to decode full 32-bit address

Name	Direction	Width	Description
read	Input	1	High when a read operation is to be performed
write	Input	1	High when a write operation is to be performed
readdata	Output	32	32-bit data to be read
writedata	Input	32	32-bit data to be written
address	Input	4	Address of the read or write operation
byteenable	Input	4	4-bit active high signal to identify which byte(s) are being written
chipselect	Input	1	High during a read or write operation

Read/Write Timing

- We will use the Avalon-MM bus with **fixed** wait-states
- Number of wait states will be 0*
- Note: timing diagram has wait state of 1! What would same thing with wait state = 0 look like?





Components with zero wait-states are allowed. However, components with zero wait-states may decrease the achievable frequency. Zero wait-states require the component to generate the response in the same cycle that the request was presented.

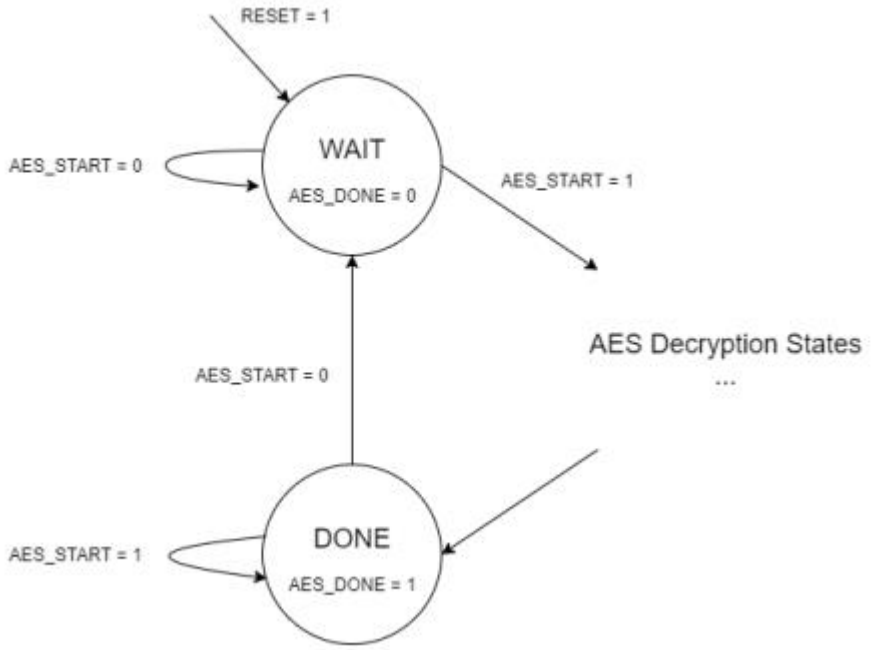
Byte Enable Support

- Have to support byte enable on Avalon writes (to FPGA registers)
- Recommended design:

byteenable[3:0]	Write Action
1111	Write full 32-bits
1100	Write the two upper bytes
0011	Write the two lower bytes
1000	Write byte 3 only
0100	Write byte 2 only
0010	Write byte 1 only
0001	Write byte 0 only

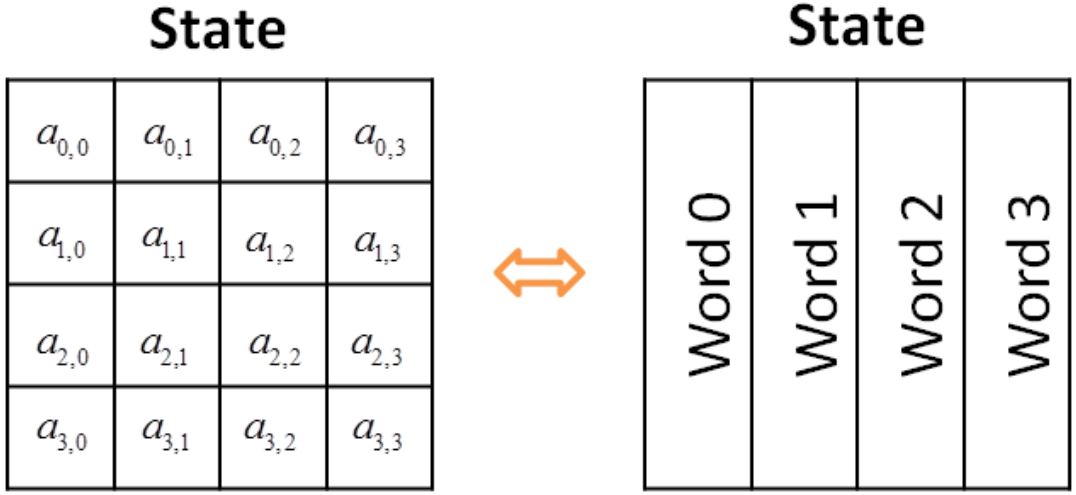
AES_Start and AES_Done registers

- AES_Start and AES_Done use only last bit of respective registers
- Follow the following state machine to start and stop the decryption process



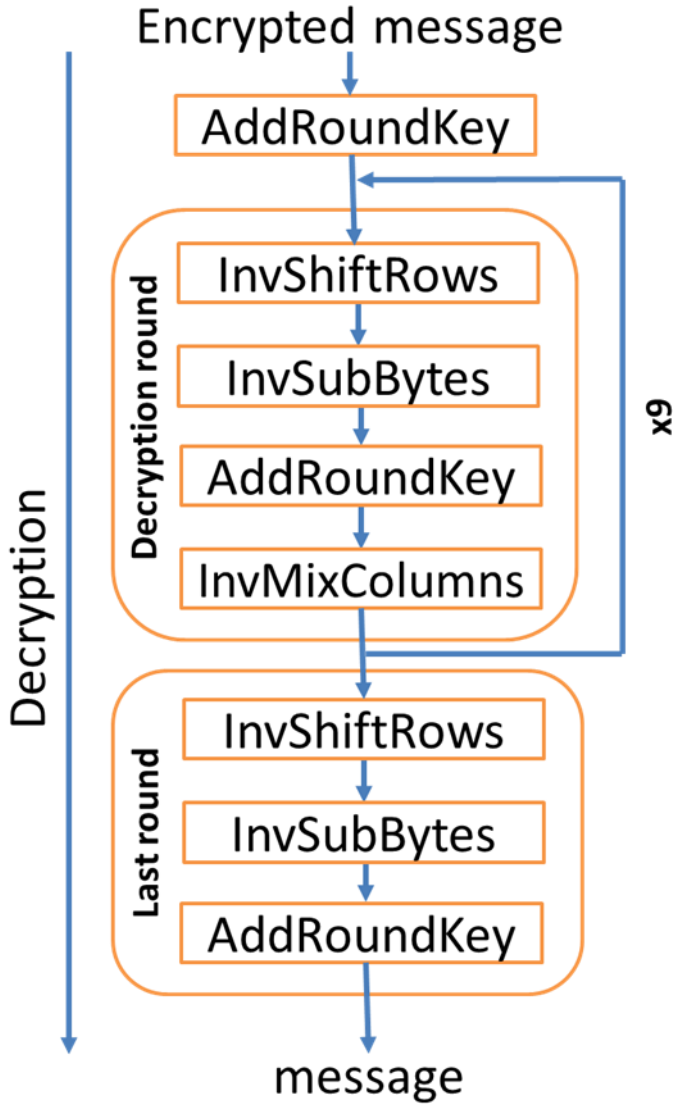
Part I: AES (128-bit)

- State – 128-bit intermediate results during the AES algorithm, arranged in column major 4x4 Bytes
- Word –the 4-Byte data from a single State column.
- Round Key – 128-bit keys derived from the Cipher Key using the Key Expansion routine. It is applied in different stages of the algorithm
- Key Schedule – 11x128-bit Round Keys derived from the Cipher Key using the Key Expansion routine



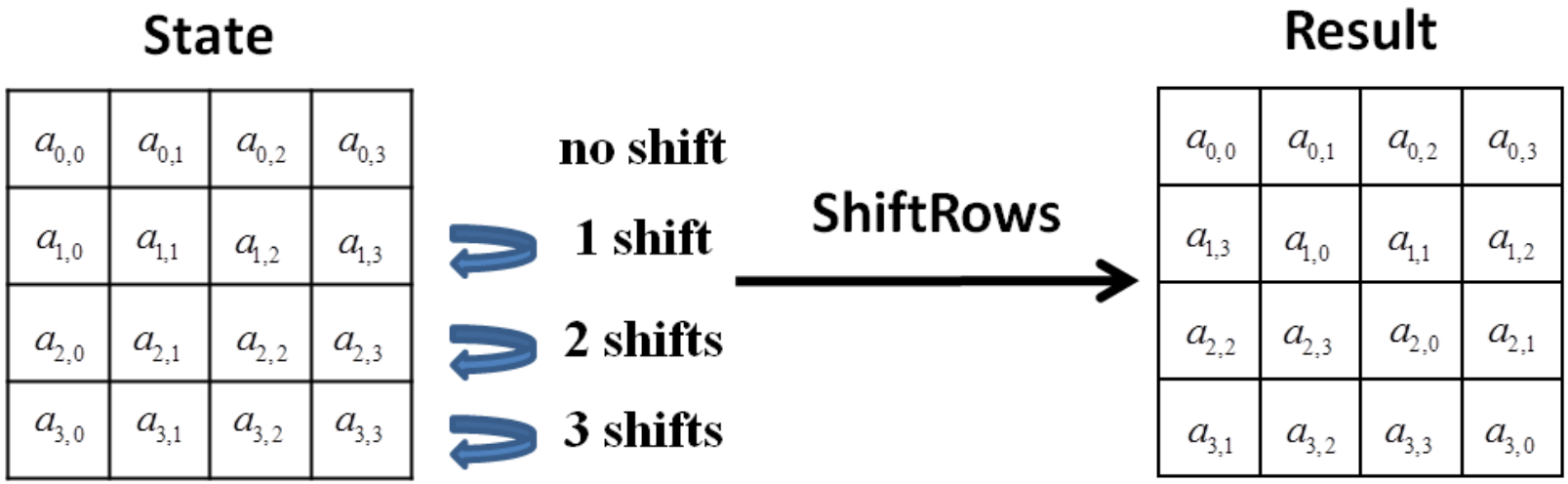
AES Decryption Algorithm

- An AES decryption goes through several rounds as well
  - Similar module routines within each round
  - Ordering of the routines are slightly different
  - 10 rounds for 128-bit AES, with 9 full rounds and a reduced last round
  - Round Keys in AddRoundKey() are generated separately from the Cipher Key



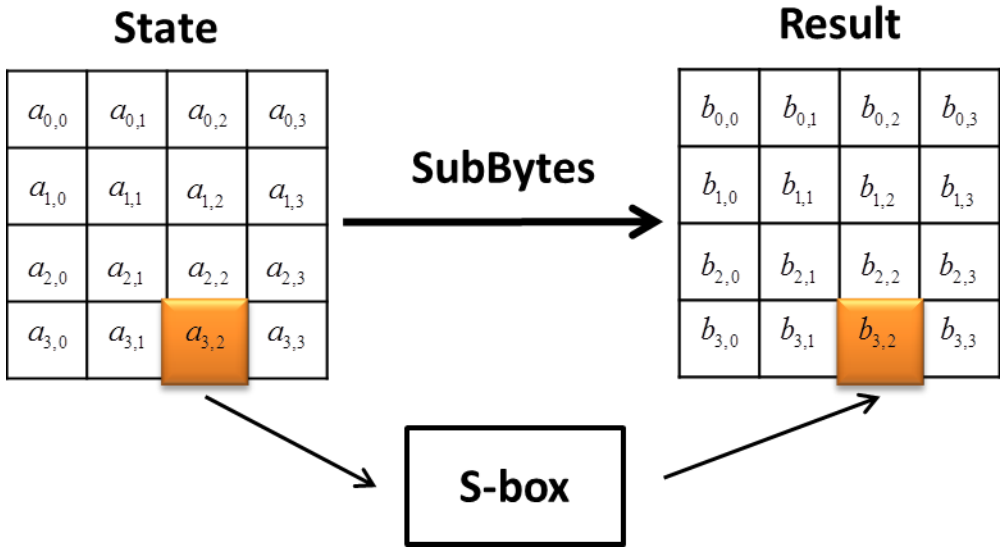
InvShiftRows ()

- InvShiftRows performs circular **right** shift
  - First row remains unchanged
  - Second row is right-circularly shifted by 1 Byte
  - Third row is right-circularly shifted by 2 Bytes
  - Fourth row is right-circularly shifted by 3 Bytes



InvSubBytes()

- InvSubBytes performs transformation in the Rijndael's finite field
  - First find the multiplicative inverse of each Byte
  - Then use an affine transformation in GF(2^8) to obtain the final value
  - This process is usually pre-computed and stored in Rijndael's S-box (substitution box) as a lookup table



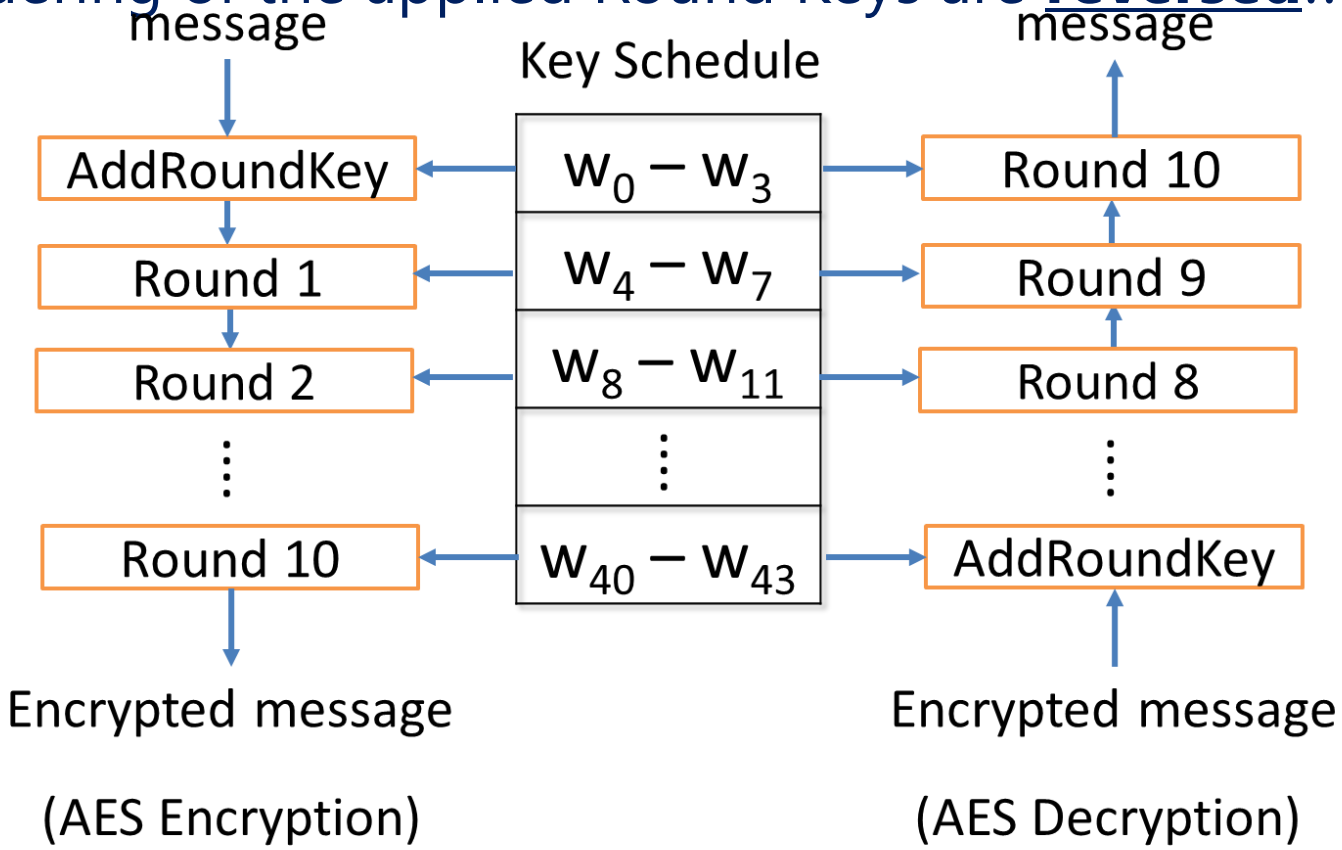
SubBytes()

- InvSubBytes performs transformation in the Rijndael's finite field
  - Identical to the SubBytes module, except using the **inverse** S-box

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1x	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2x	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3x	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4x	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5x	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6x	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7x	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8x	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9x	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
ax	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
bx	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
cx	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
dx	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
ex	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
fx	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

InvAddRoundKey ()

- XORs each Byte with the corresponding Byte from the current RoundKey
- The ordering of the applied Round Keys are **reversed**!!



InvMixColumns()

- Multiply each column by matrix as shown in **GF(2^8)**

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

- MixColumns performs matrix multiplication with each Word w_i = $\{a_{0,i}, a_{1,i}, a_{2,i}, a_{3,i}\}^T$ under Rijndael's finite field
  - {*09* }, {*0b* }, {*0d* }, and {*0e* } can be implemented using recursively – this is the approach used in the predefined InvMixColumns module

InvMixColumns()

- Provided lookup table from week 1:

```
// - This table stores pre-calculated values for all possible GF(2^8) calculations.This
// table is only used by the (Inv)MixColumns steps.
// USAGE: The second index (column) is the coefficient of multiplication. Only 7 different
// coefficients are used: 0x01, 0x02, 0x03, 0x09, 0x0b, 0x0d, 0x0e, but multiplication by
// 1 is negligible leaving only 6 coefficients. Each column of the table is devoted to one
// of these coefficients, in the ascending order of value, from values 0x00 to 0xFF.
// (Columns are listed double-wide to conserve vertical space.)
uchar gf_mul[256][6] = {
      {0x00,0x00,0x00,0x00,0x00,0x00},{0x02,0x03,0x09,0x0b,0x0d,0x0e},
      {0x04,0x06,0x12,0x16,0x1a,0x1c},{0x06,0x05,0x1b,0x1d,0x17,0x12},
      {0x08,0x0c,0x24,0x2c,0x34,0x38},{0x0a,0x0f,0x2d,0x27,0x39,0x36},
      {0x0c,0x0a,0x36,0x3a,0x2e,0x24},{0x0e,0x09,0x3f,0x31,0x23,0x2a},
      {0x10,0x18,0x48,0x58,0x68,0x70},{0x12,0x1b,0x41,0x53,0x65,0x7e},
      {0x14,0x1e,0x5a,0x4e,0x72,0x6c},{0x16,0x1d,0x53,0x45,0x7f,0x62},
      {0x18,0x14,0x6c,0x74,0x5c,0x48},{0x1a,0x17,0x65,0x7f,0x51,0x46},
```

- How would we use for GF(2^8) multiplication? E.g. 6*3

InvMixColumns()

- Example InvMixColumn():

$$(\{04\} \bullet a_{0,i}) = (\{02\} \bullet (\{02\} \bullet a_{0,i}))$$
$$(\{08\} \bullet a_{0,i}) = (\{02\} \bullet (\{04\} \bullet a_{0,i}))$$
$$(\{09\} \bullet a_{0,i}) = (\{08\} \bullet a_{0,i}) \oplus a_{0,i}$$
$$(\{0b\} \bullet a_{0,i}) = (\{08\} \bullet a_{0,i}) \oplus (\{02\} \bullet a_{0,i}) \oplus a_{0,i}$$
$$(\{0d\} \bullet a_{0,i}) = (\{08\} \bullet a_{0,i}) \oplus (\{04\} \bullet a_{0,i}) \oplus a_{0,i}$$
$$(\{0e\} \bullet a_{0,i}) = (\{08\} \bullet a_{0,i}) \oplus (\{04\} \bullet a_{0,i}) \oplus (\{02\} \bullet a_{0,i})$$

KeyExpansion()

- KeyExpansion generates a RoundKey at a time based on the previous RoundKey (use the Cipher Key to generate the first RoundKey)
  - RotWord() – circularly shift each Byte in a Word up by 1 Byte
  - SubWord() – identical to SubBytes()
  - Rcon() – xor the Word with the corresponding Word from the Rcon lookup table
- KeyExpansion hardware module is provided, but needs to wait some number of cycles in order to finish **(this is important, especially in simulation when you start simulating at time t = 0 – expanded key may not be ready!)**

for every Word w_i in all n RoundKeys ($i=1,2,\ldots,4n$, $n=10$)

$w_{temp} = w_{i-1}$

if w_i is the first Word in the current RoundKey

w_{temp} = SubWord(RotWord(w_{temp})) **xor** Rcon_n

for every Word in the current RoundKey, including the first Word

$w_i = w_{i-4}$ **xor** w_{temp}

Overall Design for Week 2

Overall Design for Week 2

- Design must only instantiate **one** each of InvMixColumns and KeyExpansion
- May instantiate multiple InvSubBytes modules
- Need to write additional modules (rotation, add round key)
- Need to design state machine which feeds entire state (128 bits) through modules (this will take many cycles, since can only have one of each module)
- No credit will be given if modules are instantiated more than once
- Pipelining can be used to improve performance, but is not necessary for full credit
- Decryption state machine should assert AES ready when decrypted message is ready to transmit back to software