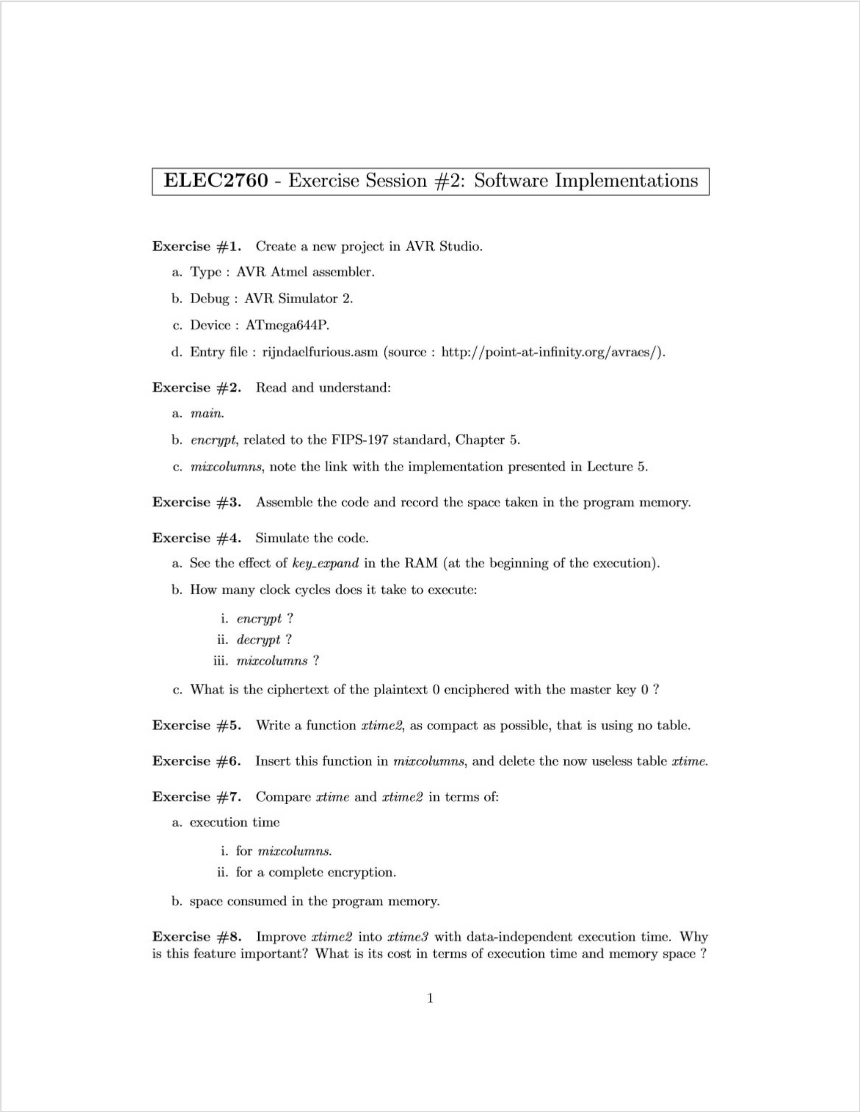
E2

04 March 2025 14:49

E2



ELEC2760 - Exercise Session #2: Software Implementations

Exercise #1. Create a new project in AVR Studio.

a. Type : AVR Atmel assembler.

b. Debug : AVR Simulator 2.

c. Device : ATmega644P.

d. Entry file : rijndaelfurious.asm (source : http://point-at-infinity.org/avraes/).

Exercise #2. Read and understand:

a. *main*.

b. *encrypt*, related to the FIPS-197 standard, Chapter 5.

c. *mixcolumns*, note the link with the implementation presented in Lecture 5.

Exercise #3. Assemble the code and record the space taken in the program memory.

Exercise #4. Simulate the code.

a. See the effect of *key_expand* in the RAM (at the beginning of the execution).

b. How many clock cycles does it take to execute:

i. *encrypt* ?

ii. *decrypt* ?

iii. *mixcolumns* ?

c. What is the ciphertext of the plaintext 0 enciphered with the master key 0 ?

Exercise #5. Write a function *xtime2*, as compact as possible, that is using no table.

Exercise #6. Insert this function in *mixcolumns*, and delete the now useless table *xtime*.

Exercise #7. Compare *xtime* and *xtime2* in terms of:

a. execution time

i. for *mixcolumns*.

ii. for a complete encryption.

b. space consumed in the program memory.

Exercise #8. Improve *xtime2* into *xtime3* with data-independent execution time. Why is this feature important? What is its cost in terms of execution time and memory space ?

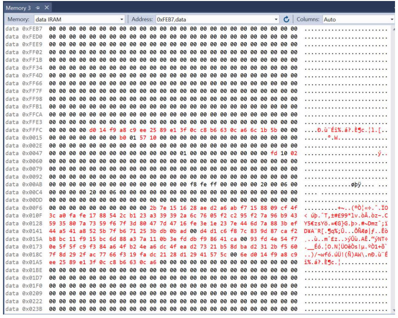
1

1) Donne
2) Understanding
   a. Main :
      i. Init stack
      ii. Load key
      iii. Expand key
      iv. Load plaintext
      v. Encrypt
      vi. Decrypt
   b. Encrypt:
      i. N rounds
      ii. Add round key
      iii. SubBytes
      iv. Shift rows
      v. S-box (in function)
      vi. Add round key (if last op)
      vii. Mix columns
   c. Mixcolumns:
      i. Move from registers, then xor it, using the matrix and LUT
3) Assembly

```
"ATmega644P" memory use summary [bytes]:
Segment  Begin    End      Code  Data  Used  Size  Use%
---------------------------------------------------------------
[.cseg] 0x000000 0x001300   802   800  1602  65536  2.4%
[.dseg] 0x000100 0x000100     0     0     0   4096  0.0%
[.eseg] 0x000000 0x000000     0     0     0   2048  0.0%
Assembly complete, 0 errors. 0 warnings
```

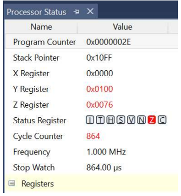
1.602 KB used

4) Simulation
   a. Key expansion (in main)
      i.



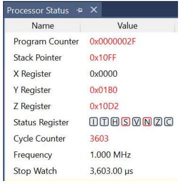
```
data 0xFEB7  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFED0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFEE9  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF02  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF1B  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF34  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF4D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF66  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF7F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFF98  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFFB1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFFCA  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFFE3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0xFFFC  00 00 00 00 d0 14 f9 a8 c9 ee 25 89 e1 3f 0c c8 b6 63 0c a6 6c 1b 5b 00 00  ....Ð.ù¨Éî%.á?.È¶c.¦l.[..
data 0x0015  00 00 00 00 00 00 00 b0 01 57 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .......°.W...............
data 0x002E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x0047  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 fd 10 02  ......................ý..
data 0x0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x0079  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x0092  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x00AB  00 00 00 00 00 00 00 00 00 00 00 00 00 00 f8 fe ff 00 00 00 00 20 00 06 00  ..............øþÿ.... ...
data 0x00C4  00 00 00 00 20 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .... ....................
data 0x00DD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x00F6  00 00 00 00 00 00 00 00 00 00 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f  ..........+~..(®Ò¦«÷.ˆ.ÏO
data 0x010F  3c a0 fa fe 17 88 54 2c b1 23 a3 39 39 2a 6c 76 05 f2 c2 95 f2 7a 96 b9 43  < úþ.ˆT,±#£99*lv.òÂ.òz–.C
data 0x0128  59 35 80 7a 73 59 f6 7f 3d 80 47 7d 47 16 fe 3e 1e 23 7e 44 6d 7a 88 3b ef  Y5€zsYö.=€G}G.þ>.#~Dmzˆ;ï
data 0x0141  44 a5 41 a8 52 5b 7f b6 71 25 3b db 0b ad 00 d4 d1 c6 f8 7c 83 9d 87 ca f2  D¥A¨R[.¶q%;Û...ÔÑÆø|ƒ..Êò
data 0x015A  b8 bc 11 f9 15 bc 6d 88 a3 7a 11 0b 3e fd db f9 86 41 ca 00 93 fd 4e 54 f7  ¸¼.ù.¼mˆ£z..>ýÛù.AÊ.“ýNT÷
data 0x0173  0e 5f 5f c9 f3 84 a6 4f b2 4e a6 dc 4f ea d2 73 21 b5 8d ba d2 31 2b f5 60  .__Éó.¦O.N¦ÜOêÒs!µ.ºÒ1+õ`
data 0x018C  7f 8d 29 2f ac 77 66 f3 19 fa dc 21 28 d1 29 41 57 5c 00 6e d0 14 f9 a8 c9  ..)/¬wfó.úÜ!(Ñ)AW\.nÐ.ù¨É
data 0x01A5  ee 25 89 e1 3f 0c c8 b6 63 0c a6 00 00 00 00 00 00 00 00 00 00 00 00 00 00  î%.á?.È¶c.¦..............
data 0x01BE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x01D7  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x01F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x0209  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x0222  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
data 0x023B  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .........................
```

      ii. Make 2 brake points, and press play, then continue to see the red changes
      iii. Key expansion -> Create the sub-keys for the rest of the algorithm
   b. Clock cycles
      (You have to turn on the Processor Status thing in debug mode, and add breaks)
      i. Encrypt :
         1) Entry of encrypt :
         2)



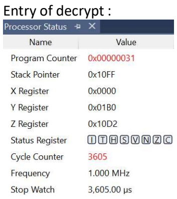
Name	Value
Program Counter	0x0000002E
Stack Pointer	0x10FF
X Register	0x0000
Y Register	0x0100
Z Register	0x0076
Status Register	I T H S V N Z C
Cycle Counter	864
Frequency	1.000 MHz
Stop Watch	864.00 µs
⊟ Registers	

         3) After Encrypt:
         4)



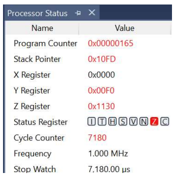
Name	Value
Program Counter	0x0000002F
Stack Pointer	0x10FF
X Register	0x0000
Y Register	0x01B0
Z Register	0x10D2
Status Register	I T H S V N Z C
Cycle Counter	3603
Frequency	1.000 MHz
Stop Watch	3,603.00 µs

         5) Total cycle count : 3603 - 864 = 2739 cycles
      ii. Decrypt :
         1) Entry of decrypt :
         2)



Name	Value
Program Counter	0x00000031
Stack Pointer	0x10FF
X Register	0x0000
Y Register	0x01B0
Z Register	0x10D2
Status Register	I T H S V N Z C
Cycle Counter	3605
Frequency	1.000 MHz
Stop Watch	3,605.00 µs

         3) Output of decrypt (at ret) :

4) 

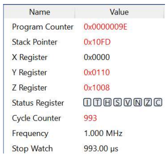
Processor Status

Name	Value
Program Counter	0x00000165
Stack Pointer	0x10FD
X Register	0x0000
Y Register	0x00F0
Z Register	0x1130
Status Register	I T H S V N Z C
Cycle Counter	7180
Frequency	1.000 MHz
Stop Watch	7,180.00 µs

5) Total cycle count : 7180 - 3605 = 3575 cycles

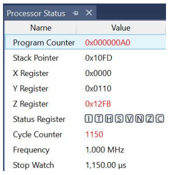
iii. Mixcolumns :

1) Before the mixcolumn:

2) 

Name	Value
Program Counter	0x0000009E
Stack Pointer	0x10FD
X Register	0x0000
Y Register	0x0110
Z Register	0x1008
Status Register	I T H S V N Z C
Cycle Counter	993
Frequency	1.000 MHz
Stop Watch	993.00 µs

3) After the mixcolumns rcall :

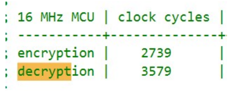
4) 

Processor Status

Name	Value
Program Counter	0x000000A0
Stack Pointer	0x10FD
X Register	0x0000
Y Register	0x0110
Z Register	0x12FB
Status Register	I T H S V N Z C
Cycle Counter	1150
Frequency	1.000 MHz
Stop Watch	1,150.00 µs

5) Total cycle count : 1150 - 993 = 157 cycles

iv. Comparaison with theory :

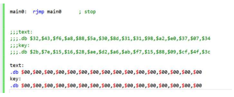
1) 

```
; 16 MHz MCU | clock cycles |
; -----------+--------------+
; encryption |     2739     |
; decryption |     3579     |
```

2) Difference for decryption since I had to measure at ret, instead of after it

c. Ciphertext of the plaintext 0 with master key 0 ?

i. Modifications :

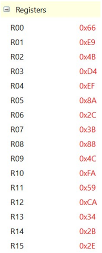
ii. 

```
main0:  rjmp main0      ; stop


;;;text:
;;;.db $32,$43,$f6,$a8,$88,$5a,$30,$8d,$31,$31,$98,$a2,$e0,$37,$07,$34
;;;key:
;;;.db $2b,$7e,$15,$16,$28,$ae,$d2,$a6,$ab,$f7,$15,$88,$09,$cf,$4f,$3c

text:
.db $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
key:
.db $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
```

iii. CT :

1) 

⊟ Registers

R00	0x66
R01	0xE9
R02	0x4B
R03	0xD4
R04	0xEF
R05	0x8A
R06	0x2C
R07	0x3B
R08	0x88
R09	0x4C
R10	0xFA
R11	0x59
R12	0xCA
R13	0x34
R14	0x2B
R15	0x2E

Exercise #5. Write a function *xtime2*, as compact as possible, that is using no table.

Exercise #6. Insert this function in *mixcolumns*, and delete the now useless table *xtime*.

Exercise #7. Compare *xtime* and *xtime2* in terms of:

a. execution time

i. for *mixcolumns*.

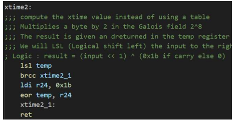
ii. for a complete encryption.

b. space consumed in the program memory.

Exercise #8. Improve *xtime2* into *xtime3* with data-independent execution time. Why is this feature important? What is its cost in terms of execution time and memory space ?

Warning, the xtim2 and xtime3 functions need to be above xtime to not be outside the address space

5) Xtime2 as compact as possible

a. 

```
xtime2:
;;; compute the xtime value instead of using a table
;;; Multiplies a byte by 2 in the Galois field 2^8
;;; The result is given an dreturned in the temp register
;;; We will LSL (Logical shift left) the input to the righ
; Logic : result = (input << 1) ^ (0x1b if carry else 0)
    lsl temp
    brcc xtime2_1
    ldi r24, 0x1b
    eor temp, r24
    xtime2_1:
    ret
```

6) Insert in mix columns

a. ?

7) Compare Xtime and Xtime2 in terms of :

a. Execution time

i. Mixcolumns

1)

Xtime	157
Xtime2	1241 - 994 = 247

ii. Complete encryption

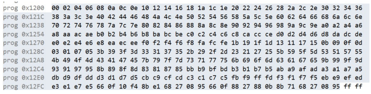
1)

Xtime	2739
Xtime2	4485 - 864 = 3621

b. Space consumed in the program memory

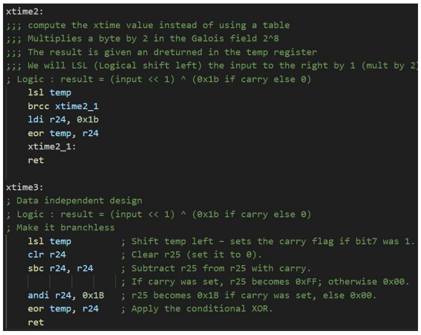
i.

Xtime	16x16=256 B
Xtime2	10 B (Xtime2 and Xtime3 are in the same address 0x130A

ii. 

```
prog 0x1200  00 02 04 06 08 0a 0c 0e 10 12 14 16 18 1a 1c 1e 20 22 24 26 28 2a 2c 2e 30 32 34 36
prog 0x121C  38 3a 3c 3e 40 42 44 46 48 4a 4c 4e 50 52 54 56 58 5a 5c 5e 60 62 64 66 68 6a 6c 6e
prog 0x1238  70 72 74 76 78 7a 7c 7e 80 82 84 86 88 8a 8c 8e 90 92 94 96 98 9a 9c 9e a0 a2 a4 a6
prog 0x1254  a8 aa ac ae b0 b2 b4 b6 b8 ba bc be c0 c2 c4 c6 c8 ca cc ce d0 d2 d4 d6 d8 da dc de
prog 0x1270  e0 e2 e4 e6 e8 ea ec ee f0 f2 f4 f6 f8 fa fc fe 1b 19 1f 1d 13 11 17 15 0b 09 0f 0d
prog 0x128C  03 01 07 05 3b 39 3f 3d 33 31 37 35 2b 29 2f 2d 23 21 27 25 5b 59 5f 5d 53 51 57 55
prog 0x12A8  4b 49 4f 4d 43 41 47 45 7b 79 7f 7d 73 71 77 75 6b 69 6f 6d 63 61 67 65 9b 99 9f 9d
prog 0x12C4  93 91 97 95 8b 89 8f 8d 83 81 87 85 bb b9 bf bd b3 b1 b7 b5 ab a9 af ad a3 a1 a7 a5
prog 0x12E0  db d9 df dd d3 d1 d7 d5 cb c9 cf cd c3 c1 c7 c5 fb f9 ff fd f3 f1 f7 f5 eb e9 ef ed
prog 0x12FC  e3 e1 e7 e5 66 0f 10 f4 8b e1 68 27 08 95 66 0f 88 27 88 0b 8b 71 68 27 08 95 ff ff
```

8) Xtime3 be branchless

a. 

```
xtime2:
;;; compute the xtime value instead of using a table
;;; Multiplies a byte by 2 in the Galois field 2^8
;;; The result is given an dreturned in the temp register
;;; We will LSL (Logical shift left) the input to the right by 1 (mult by 2
; Logic : result = (input << 1) ^ (0x1b if carry else 0)
    lsl temp
    brcc xtime2_1
    ldi r24, 0x1b
    eor temp, r24
    xtime2_1:
    ret

xtime3:
; Data independent design
; Logic : result = (input << 1) ^ (0x1b if carry else 0)
; Make it branchless
    lsl temp          ; Shift temp left - sets the carry flag if bit7 was 1.
    clr r24           ; Clear r25 (set it to 0).
    sbc r24, r24      ; Subtract r25 from r25 with carry.
                      ; If carry was set, r25 becomes 0xFF; otherwise 0x00.
    andi r24, 0x1B    ; r25 becomes 0x1B if carry was set, else 0x00.
    eor temp, r24     ; Apply the conditional XOR.
    ret
```

b. Time :

i. Encryption : 4710 - 864 = 3846

ii. MixColumns : 1273 - 994 = 279

c. Memory :

   i. Same : 10 B

d. ITS IMPORTANT SO THAT THE POWER DOES NOT LEAK THE DATA