Brussels Faculty of Enginering

Université Libre de Bruxelles

Vrije Universiteit van Brussel

Academic Year 2016-2017

Project Report

Encrypting messages using AVR Assembly "EnigmAssembly"

Cédric Hannotier, Mathieu Petitjean

ELEC–Y418: Sensors and Microsystem electronics

1 Introduction

This report presents the implementation of XOR cipher on a ATmega328P microcontroller using AVR Assembly: the EnigmAssembly. It can detect the inputs of a keyboard, display any ASCII character and encrypt them using a XOR cipher. By typing the correct password again, a message can be retrieved back after being encrypted.

First, a simple user guide describing the human-machine interface is provided and details the usage of the EnigmAssembly. Next, the encryption scheme is presented and the security of the implementation is discussed. Finally, the functioning of the code is explained.

2 User Guide

To encrypt a message and decrypt it back, the user must perform the following steps:

1. **Message encoding:** The first step is to type the message that must be encoded using the keyboard. This keyboard implements the ASCII table: any character of the table (shown in Figure 1) from code `0x20` to `0x7F` can be displayed on the screen by successively typing the two hexadecimal numbers composing the code. For example, if one wants to input the character A, which ASCII code is `0x41`, one needs to type 4 then 1. The LED labelled `PC2` will be lit when only the first of the hexadecimal numbers has been recorded.

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Figure 1: ASCII table

Any hexadecimal code not displayable or non existing is represented by a dot.

Two ASCII characters of the first column have been implemented. The character "backspace" (`0x08`) allows to delete the last entry if a mistake occurred while typing. The character "end of text" (`0x03`) must be typed when the message is fully encoded. It signifies to the EnigmAssembly that the next step begins: the encryption.

2. **Message encryption:** Once the "end of text" is typed, the message will be encrypted using a password. The password needs to be at least as long as the input message for the all the letters to be encrypted (two hexadecimal numbers per letter) but can be longer. It can consist of a repetition of the same sequence but doing so limits the security level. For example, to encrypt the character A with the password `0x12`, one needs to successively type 4, 1 (encode A), 0, 3 (end of file), 1, 2 (password). The result should be the display of the ciphertext S.

   The message is now safely encrypted, and the decryption step begins when the joystick is pressed. If something went wrong when typing the password, it is possible to go back to step 1 by typing the "end of file" character.

3. **Message decryption:** When the joystick is pressed, the LED labelled `PC3` is lit to indicate that the EnigmAssembly is in decryption mode. The initial message is retrieved only if the correct password is used.

   The process is then exactly the same as in the encryption step, but the message will be decrypted as the password is typed instead of encrypted. After pressing the joystick, the character A will appear again if 1 then 2 is pressed.

   The EnigmAssembly can be used to encode a new message and start over at step 1 by pressing the joystick again.

3 Encryption algorithm

The EnigmAssembly implements a so-called XOR cipher. Its functioning is based on the fact that applying a bitwise exclusive OR (XOR) operator twice to a word gives the word back. Indeed, by denoting the bitwise XOR by $\oplus$:

$$\underbrace{(M \oplus K)}_{C} \oplus K = M \oplus 0 = M$$

If a XOR is applied to the message M with the key K, the result is the encrypted message C. The message is found back when applying a XOR with the same key. It can be seen as a digital equivalent to the symmetric Vigenère cipher since each character is encrypted using a different substitution rule.

Maximal security is obtained when the password is as long as the message and does not need to be repeated and when the password is never reused. The encryption scheme is then a One-Time-Pad (OTP) and is unbreakable to any other attack than the brute-force if the key is random and the plaintext unknown.

Of course, such a scheme has drawbacks as it requires that the transmitter and receiver share the knowledge of the key. Furthermore, because $M \oplus C = K$, the key can be deduced from the knowledge of the initial message and the encrypted message. Assymetric key cryptography (or public key cryptography) is widely used today, but the algorithms used, such as RSA, are heavy on computations. An often used workaround is the establishment of a secure connection via RSA during which a symmetric key can be setup for later communications.

From the implementation point of view, care has been taken to the fact that after the encryption, neither the initial message nor the password is saved. Even if an attacker has full access to the content of the registers or the memory of the EnigmAssembly, the only way to get the initial message back is by typing the correct password.

4 Implementation

It is first worth mentioning that the code does not contain any functions or PUSH/POP instructions. This is totally intentional. Going to Assembly to code a microcontroller allows a more precise control of the memories and the registers, and is done in a desire to have strong optimisation. In this mindset, everything was done to reduce as much as possible the overhead and the access to memories that are not registers (and hence slower, requiring more clock cycles). The status register is (and can) thus not be used in this configuration.

The program uses one 8-bit timer which triggers an interrupt routine when it overflows. This interrupt routine handles the refresh of the screen display at a rate of around 500Hz (roughly 60Hz per line since the lines are displayed one by one), as shown in Figure 2. The delay of 100 µs is needed to have a sufficient brightness in the display, but does not hinder the functioning of the main loop, which handles the keyboard and XOR operation. Note that the $\frac{1}{500}$ s interval before the next overflow is much greater than the time taken by the interrupt routine but this was not represented as such on the Figure for the sake of clarity.

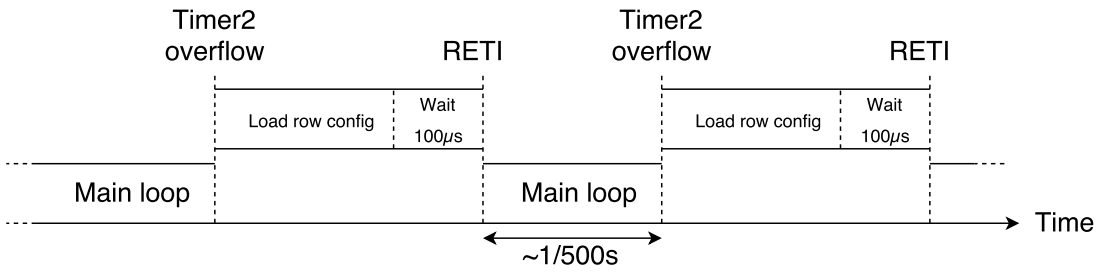

Figure 2: Sequential execution of the program

In order to implement the XOR operation on the stored message, all the characters of the ASCII table are stored into memory. Knowing the address of the first character, the ASCII character of code `0x09` is stored at a memory emplacement `0x12` (the double of the ASCII offset) further than the first one. Each character is then represented by its offset with respect to the first one via a system of "pointers of pointers". Performing a XOR on these offsets is equivalent to perform a XOR on the actual ASCII codes.

The structure of the main loop is represented in Figure 3. Registers are used as memories to remember the state of the joystick (pressed or not) and the operating mode of the program (write a message or perform a XOR). The T flag is used to determine if a key was pressed at the previous iteration.

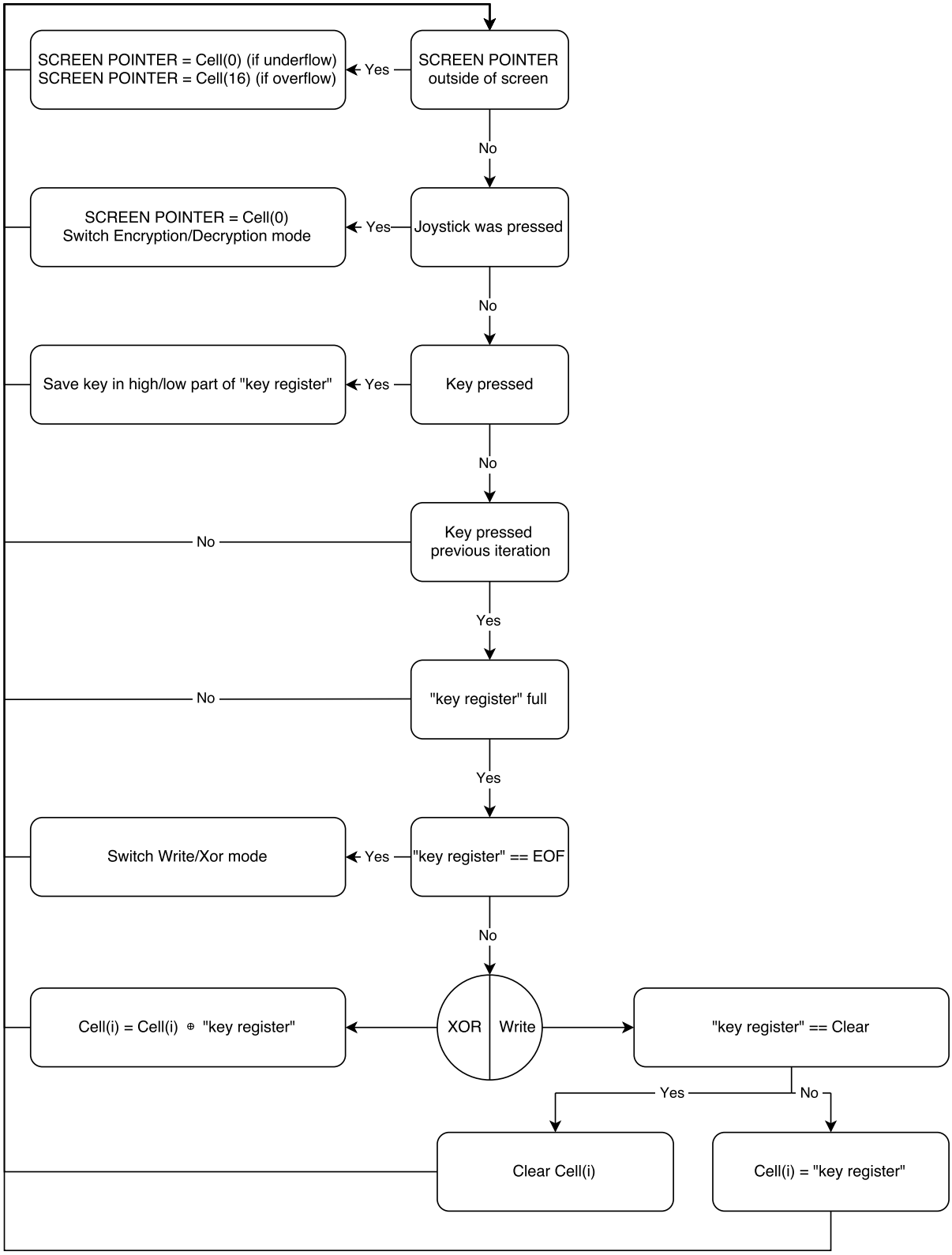

Figure 3: Main loop structure