

COMP3211/9211 Computer Architecture 2011 Project

Encryption/Tagging Processor Design

This project is to be completed in teams of 3 people drawn from your Tute/Lab class. Deliverables for the part 1 and part 2 of the project contribute to the group report due on Friday, 27 May (week 12).

Problem Overview

The goal of this project is to design an application-specific processor to perform encryption and generate a tag for an input text string, as depicted in Figure 1.

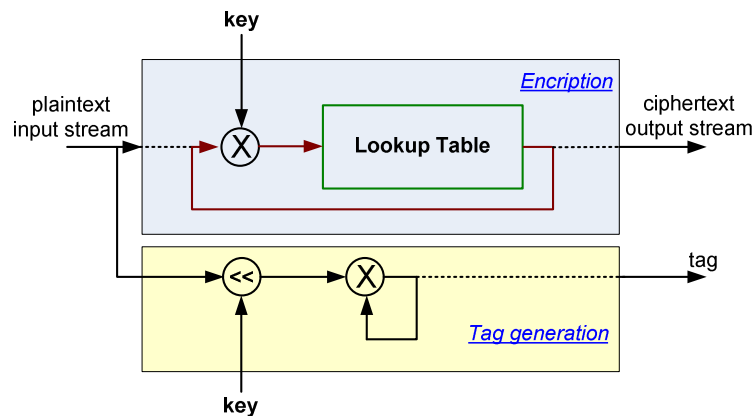


Figure 1: Encryption and Tag Generation System

The system takes a plaintext stream as input and converts it into a ciphertext stream, and at the same time generates a tag for the plaintext. The plaintext is in 8-bit ASCII format (the ASCII table is attached at the end of this document). The encryption key is 64 bits, which is divided into 8 sub-keys, 8 bits each. Both encryption and tag generation are elaborated below.

Encryption

To encrypt a character, the ASCII value of the input character is first XOR'ed with the first sub-key, the result of which is used as the address to find a value in the lookup table. The lookup table value (also in 8 bits) together with the second sub-key is used for the next round of XOR-Lookup process. This process is repeated until all 8 sub-keys are applied. Therefore, 8 iterations of XOR-Lookup are required for each input character encryption.

To simplify the problem, we assume the lookup table values can be calculated with the following formula,

$$V(A) = (A + C)_L \parallel A_H \quad (1)$$

where A is the 8-bit address value, C a constant, and \parallel the concatenation operation; X_H denotes the high 4 bits (aka, nibble) of X , X_L the low 4 bits. Namely, each element in the table is formed by the low nibble of $(A+C)$, followed by the high nibble from address A . For example, assume $C=0x02$, the content at address 00001111 is 00010000.

As an example, Figure 2 illustrates how character 'H' is encrypted using the proposed encryption function with the encryption key 0x123456789ABCDE3F and $C=0x01$.

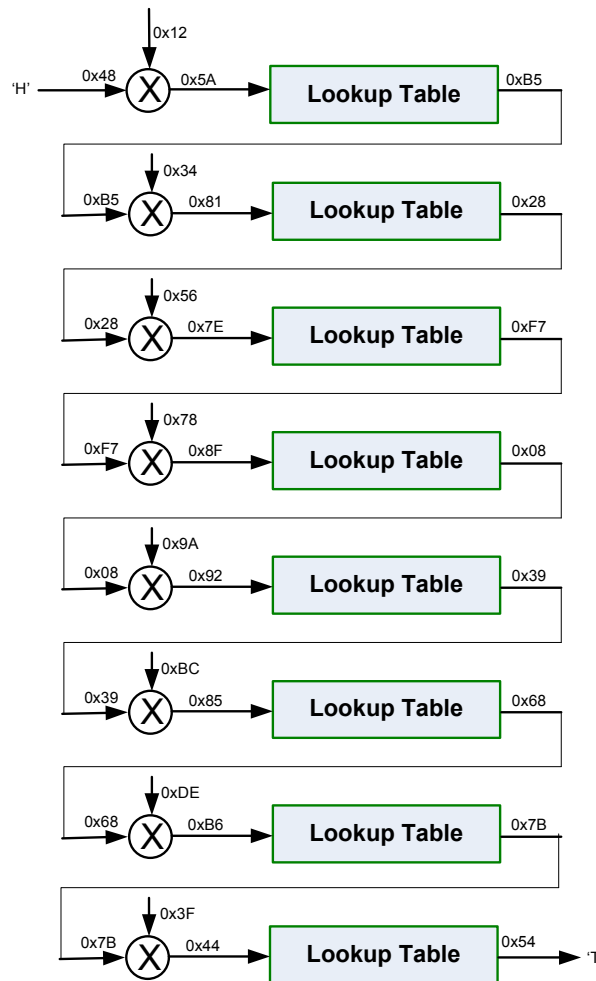


Figure 2: An Encryption Example

Tag Generation

The tag is generated by XOR'ing the shifted ASCII-values of all characters in the plaintext stream. The number of bits to be shifted for a character is determined by one bit value in the key; if the bit value is 1, then the character's ASCII value is left-shifted 1

bit, otherwise no shift when the bit value is 0. The bits of the key are used in sequence and wrapped when the last bit is reached.

Figure 3 shows an example of how the tag for string “CompArch” is generated. Here we use a short key of 4-bits, 1010, for demonstration. In your project design, you will use the same key for both encryption and tag generation.

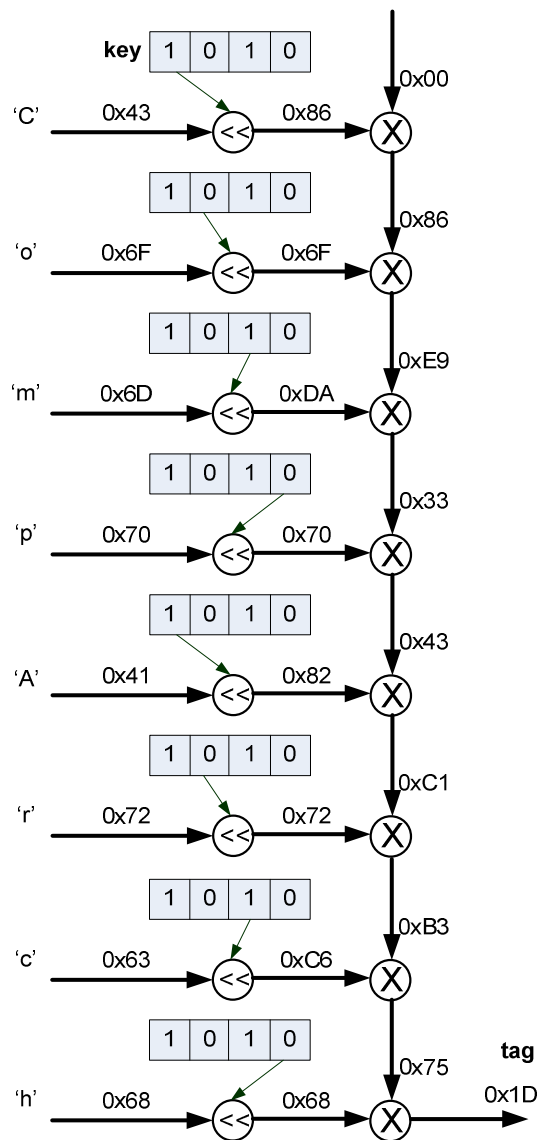


Figure 3: A Tag-generation Example

In this project, we assume the input text and the encryption key are initially stored in the memory, and the ciphertext and tag are output to the memory.

You are required to develop the following two small processors for such a system (with a given **non-zero** C ($15 > C > 0$)):

1. Single cycle processor (Project part 1)
2. Pipelined processor (Project part 2)

Project Deliverables

You are required to work in group of three people to complete the project. You are free to group as you wish. Please consider grouping together with people who aspire to achieve a similar grade as you. It is also suggested that your group has at least one student who has learnt VHDL and at least one student who is familiar with programming in a high level language.

Please ensure the name of each group member is clearly indicated on each submission and in a comment at the head of each source code file.

Each part of the project must be clearly documented in the final Group Report. The documentation for each project part should be prepared in a professional manner and each sub-part should be clearly labeled. The Group Report should contain as a minimum:

- A title, the names and student IDs of the group members, and the Tute/Lab class you belong to;
- A table of contents;
- A major section for each part;
- Components requested for each part within the relevant part.

The report should be prepared using a word processor or typesetting program and a laser-printed copy should be submitted to your tutor by the end of week 12. The detail of project report submission will be given in week 11.

Note that there is a 10% reduction in the mark for your group's report for each day your submission is delayed after the deadline.

For this project you are permitted to make use of IP (intellectual property) e.g. code, designs, diagrams, formats, layouts, etc. that you have not created under the following conditions:

Any IP you or your partner(s) have not created must clearly identify the source of the IP, be it another group in this class, a previous year's student, the internet, or from texts – any unoriginal design or code **MUST** be attributed to its source in a clearly marked comment where the material is included indicating the extent of the externally sourced material e.g. start and end of copied code – if you do not do so, your submission will be treated as having been plagiarized (see the Course Outline). Note: Templates generated by the WebPACK/ISE tools are excluded from this requirement.

The next part of the document is for part 1 of the project. The specification for part 2 will be given in week 7.

Project Part 1

The project part 1 is to be completed in teams of 3 people drawn from your Tute/Lab class. Your design, implementation and results are to be presented during your Tute/Lab class in week 6.

Objectives of Part 1

- To design an instruction set that is efficient, easy to implement, and allows the encryption function and tag generation to be completed as quickly as possible;
- To build a single-cycle processor implementation of your instruction set in VHDL;
- To simulate and verify your design by running your design for a string (e.g. your surname) on your processor core; and
- To analyze your design for performance.

Part 1 Requirements

- Each project group is required to make a detailed project plan which includes
 - Tasks that need to be performed in order to complete this part of project;
 - The schedule of the tasks;
 - The role of each member for these tasks;
 - The test method for each task; and
 - A project management strategy to ensure that the project work is carried out smoothly and completed on time with as good quality as possible.
- Your design must meet the following specific requirements:
 - It can encrypt a plain text of up to 100 characters;
 - Your VHDL model must take component delays into account. The related timing information for some functional components is as follows:
 - i. PC register propagation delay: 0.5ns
 - ii. Adder: 8-bit adder 1 ns; 16-bit adder 1.5ns;
 - iii. 8-bit ALU: 1.5ns; 16-bit ALU: 2ns;
 - iv. Memory/Register file latencies (note the delay formula is used just for easy of specification and may not true for real memories)
 1. $8N \times 8\text{-bit}$: $(0.3 + \log_2 N \times 0.4)\text{ns}$, where $N = 1, 2, 4, 8, \dots$
 2. $8N \times 16\text{-bit}$: $(0.5 + \log_2 N \times 0.5)\text{ns}$, where $N = 1, 2, 4, 8, \dots$
 - v. 2-to-1 MUX: 0.2 ns;
 - vi. Logic gates with fan-ins of 4 or below: 0.2 ns
 - vii. Control unit: 1.5 ns

The following serves as an example of some tasks (and suggestions) that you may consider for this project part.

1. Developing a C or Java high-level program for the encryption/tagging function and running the program to generate results for a given input.
2. Based on the high level program, developing an instruction set that provides the operations required by the program. Attempting to develop a simple and efficient ISA.

3. With the ISA designed in task 2, translating the high-level program into assembly code. Minimizing the number of machine instructions to be executed.
4. Constructing a single cycle processor at register transfer level with a set of functional components to implement the ISA designed in task2. You may use the skeleton project at
 ~cs3211/public_html/refs/models/single_cycle_core.zip
as a starting point for your VHDL implementation. Be sure to document your ISA design in a clear and concise manner in your VHDL source code.
5. Determining the clock cycle time of the single cycle processor.
6. Initializing the instruction and data memory with appropriate input data and simulating the encryption/tagging program (developed in task 3) on the processor with ISE and Modelsim.
7. Analyzing the simulation results to debug and verify your design.

Part 1 Presentations

You and your group are required to present your group's: a) ISA design and the assembly language program; b) the single cycle processor design; and c) the simulation results during your usual Tute/Lab class in Week 6. Each member is given 5 minutes to present one of the three design tasks. You should pace your presentation and present at an appropriate level so as to allow for one or two questions for each presentation within the allocated time frame.

All members in the group should work together to prepare for presentation slides and make your group presentation go smoothly. Each group is required to submit an electronic copy of their presentation slides (preferably in the PowerPoint format) one day before the presentation. The submission instructions will be sent to you in week 5.

Presentations will be assessed by other groups in your tute class. The score sheets for peer-assessment will be made available by week 5.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL