Present Cipher Encryption IP Core

Coded by: Reza Ameli

Digital Systems Lab

Ferdowsi University of Mashhad, Iran

http://commeng.um.ac.ir/dslab

1. Introduction

Present is a lightweight block cipher designed for hardware constrained applications such as RFID tags.

This cipher is an example of SPN ciphers. The block size is 64 bits, key size can be either 80 or 128 bits and the number of rounds is 31.

The S-Box used in Present is a 4-bit to 4-bit S-Box which is invoked both in the substitution layer and in the key scheduling routine.

This project entails an encryption-only implementation of Present cipher with key size equal to 80 bits.

The design is iterative that is the same module of one encryption round will be used 31 times. This resource reuse also applies to the key scheduling routine.

In each round of encryption, all 64 bits of state are passed through 16 S-Boxes (each having a width of 4). Implementation of these 16 S-Boxes can either direct (instantiating all 16 S-Boxes in parallel) or iterative (instantiating less than 16 S-Boxes and using them taking turns).

This design directly instantiates all of the 16 S-Boxes and another single S-Box for the key scheduling section.

2. Architecture & IOs

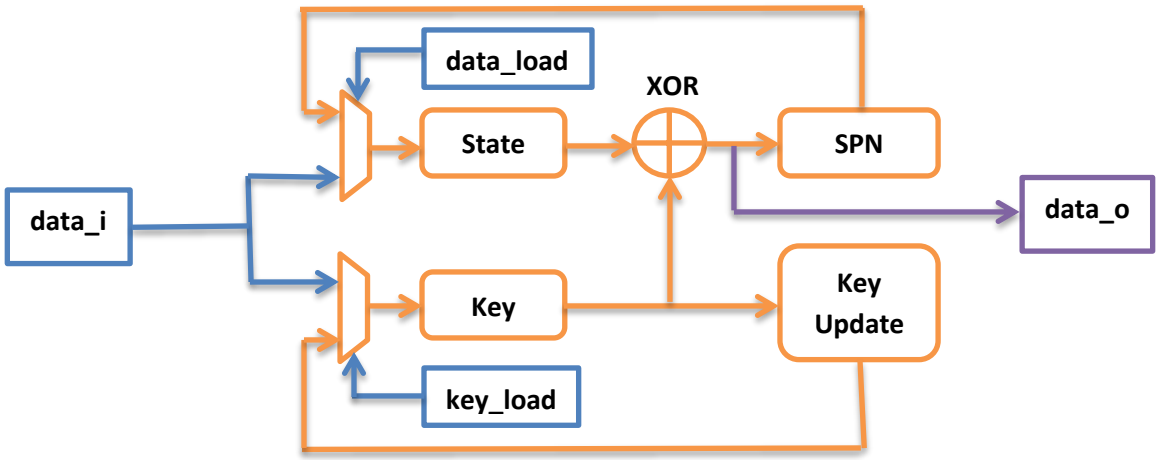
Inputs and Outputs of the design are:

Name	Direction	Width (bits)	Description
data_o	Output	64	Ciphertext will appear here
data_i	Input	80	Key and plaintext must be applied here
data_load	Input	1	'1' when loading the key
key_load	Input	1	'1' when loading the plaintext
clk_i	Input	1	Clock signal (operates at rising edge)

Internal signals and registers of design top (present_encryptor_top):

Name	Width (bits)	Type	Description
state	64	Register	Current state of cipher
round_counter	5	Register	5-bit counter (from 1 to 31)
key	80	Register	Register that holds the 80-bit key and later its updates
round_key	64	Signal	Round key that is 64 leftmost bits of key register and will be XORed by the state
sub_per_input	64	Signal	Input to Substitution-Permutation network
sub_per_output	64	Signal	Output of Substitution-Permutation network
key_update_output	80	Signal	Value that will replace the content of key register

This figure shows the architecture of design (inputs are blue, outputs are purple and internals are orange):



3. Operation

This design has two control inputs named key_load and data_load. When either of these inputs is high, at the rising edge of the clock signal (clk_i) the (required number of) bits present at data_i are copied to the corresponding register.

In case of key_load all bits of data_i (which is 80 bits wide) are copied to key register (which is also 80 bits wide).

In case of data_load, 64 rightmost bits (data_i[63:0]) of data_i are copied to state register.

One other event that happens when data is loaded (i.e. data_load = '1') is that round_counter is set to '1' (0b00001) i.e. loading a new plaintext into the state register also resets the state machine, that is why this design does not have a reset signal.

After loading the key and the plaintext, both key_load and data_load must be '0'. After setting the control inputs to '0', 30 clocks must be applied in order to data_o contain a valid ciphertext. That is after the rising edge of the 30^{th} clock signal, data_o has the correct cipher text.

So the complete encryption of plaintext requires 32 clocks.

After encrypting one plaintext and in order to encrypt another, all of the steps explained above must be done again.

Following steps briefly summarize the operation of this design:

1) Load the key (key_load = '1', data_load = '0')
2) Load the plaintext (key_load = '0', data_load = '1')
3) Set control inputs to '0' (key_load = '0', data_load = '0')
4) Apply 30 clocks