Full Hash algorithm v3 (DES S-box)

Project

Design the following hash module based on the S-box of DES algorithm. The hash algorithm generates a 32-bit digest formed by the 8 nibbles (4-bit vectors) $H[i]$, from $H[0]$ up to $H[7]$, being $H[0]$ the most significant nibble. For each message the $H[i]$ variables are initialized with the following values:

	$\boldsymbol{H[0]}$	$\boldsymbol{H[1]}$	$\boldsymbol{H[2]}$	$\boldsymbol{H[3]}$	$\boldsymbol{H[4]}$	$\boldsymbol{H[5]}$	$\boldsymbol{H[6]}$	$\boldsymbol{H[7]}$
Init. value	4'h3	4'hF	4'hA	4'h1	4'hE	4'hF	4'h2	4'h3

The, for each byte M of the input message (i.e. the 8-bit ASCII code of a message character) the hash module performs the following operations (1):

$$for\ (r = 0;\ r < 12;\ r++)$$
$$for(i = 0; i < 8; i++)$$
$$H[i] = \big(H[(i+1)\ mod\ 8] \oplus S(M_6)\big) \ll \lfloor i/2 \rfloor$$

where

$mod\ n$	is the modulo operator by n.
$\oplus$	is the XOR operator.
$X \ll n$	is the left circular shift by n bits.
$\lfloor x \rfloor$	is the floor function (applied to argument x)
M_6	is a 6-bit vector generated from M (input message byte) by the following compression function: • assuming the message byte $M\ =\ M[7:0]$ • then $M_6 = \{M[3] \oplus M[2], M[1], M[0], M[7], M[6], M[5] \oplus M[4]\}$ being $\{\}$ the concatenation operator and each $M[n]$ the nth bit of byte $M[7:0]$.
$S(\ \ )$	is the S-box transformation of DES algorithm, that works over a byte.

Once the last message byte has been processed, the hash module performs the following operations (2):

$$for(i = 0; i < 8; i++)$$
$$H[i] = \big(H[(i+1)\ mod\ 8] \oplus S(C_6[i])\big) \ll \lfloor i/2 \rfloor$$

where $C_6[i]$ is a 6-bit vector obtained from $C[i]$ and being $C[i]$ the ith byte, for $i = 0,1,2, \dots ,7$, of the counter C (64 bits). Each 6-bit vector $C_6[i]$ is obtained from the corresponding $C[i]$ byte as it follows:

- assuming the ith byte $C[i]\ =\ C[i][7:0]$
- then $C_6[i] = \{C[i][7] \oplus C[i][1], C[i][3], C[i][2], C[i][5] \oplus C[i][0], C[i][4], C[i][6]\}$

Note that C reports the real number of byte length, i.e. if the message length is 1 byte, then $C\ =\ 1$ (not 0). Therefore, it is possible to calculate also the digest of empty messages, i.e. performing only operations (2) with $H[i]$ variables equal to the initialization value and counter $C = 0$.

Additional design specifications

- The module shall have an asynchronous active-low reset port.
- The module interface should include appropriate flags to indicate the beginning and/or the end of a message, and, accordingly, the module should integrate appropriate logic for counting the bytes of the message.

- The module interface shall include an input flag to be driven as it follows: 1'b1, when input message byte on the corresponding input port is valid and stable (i.e. it can be used by the internal module logic), 1'b0, otherwise.
- The module interface shall include an output flag to be driven as it follows: 1'b1, when output digest on the corresponding output port is ready and stable (i.e., external modules can read and use it), 1'b0, otherwise.

Hints

- For DES S-box function implement the LUT version (for faster developing). Below it is reported the S-box of DES algorithm, in binary format and an example on how it works. This S-Box has 6 input bits and 4 output bits. The first and last bits are used to identify the row while the central bits are used to identify the column. For example, the input number "011011" has at its extremes the bits "01" ("**0**11011**1**") and centrally the bits "1101" ("0**1101**1") which produce as output the value "1001".

S_5		4 central bits															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
External bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1100	0011	1001	1000	0110
	10	0100	0010	0001	1011	1100	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1100	0100	0101	0011

- Testing should include verifying what happens (to the output digest) when identical messages and different messages (both of the same length and of different lengths) are processed.