Full Hash algorithm v2 (DES S-box based)

Referent

Email luca.crocetti@phd.unipi.it
Teams l.crocetti@studenti.unipi.it

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 concatenation of 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:

	$H[0]$	$H[1]$	$H[2]$	$H[3]$	$H[4]$	$H[5]$	$H[6]$	$H[7]$
Init. value	4'h4	4'hB	4'h7	4'h1	4'hD	4'hF	4'h0	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 operation:

~~$for\ (r = 0;\ r < 64;\ r + +)$~~ for (r = 0; r < 4; r++)
$for(i = 0; i < 8; i + +)$
$H[i] = \left(H[(i+1)\ mod\ 8] \oplus S(M_6)\right) \ll \lfloor i/2 \rfloor$

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 operation:

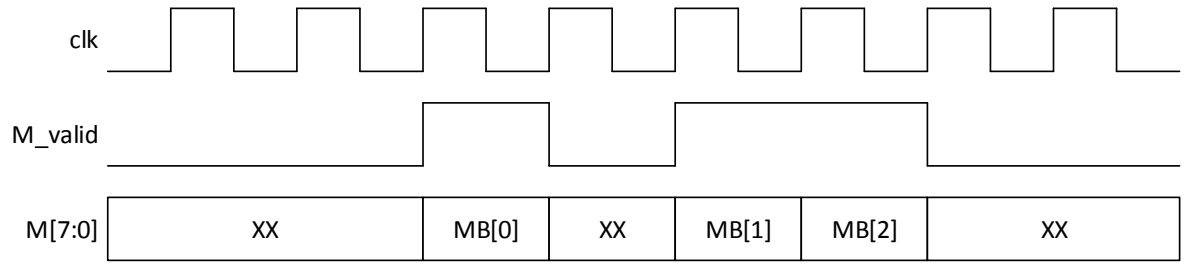
$for(i = 0; i < 8; i + +)$
$H[i] = \left(H[(i+1)\ mod\ 8] \oplus S(C_6[i])\right) \ll \lfloor i/2 \rfloor$

where $C_6[i]$ is a 6-bit vector obtained from $C[i]$ and being $C[i]$ the ith byte, for $i = 0,1,2,\ldots,7$, of the counter C (64 bits). Note the counter C which reports the (real) byte length of message, i.e. if the message length is 1 byte, then $C = 1$ (not 0) and it has to be provided as input. Each 6-bit vector $C_6[i]$ is obtained from the corresponding $C[i]$ byte as it follows:

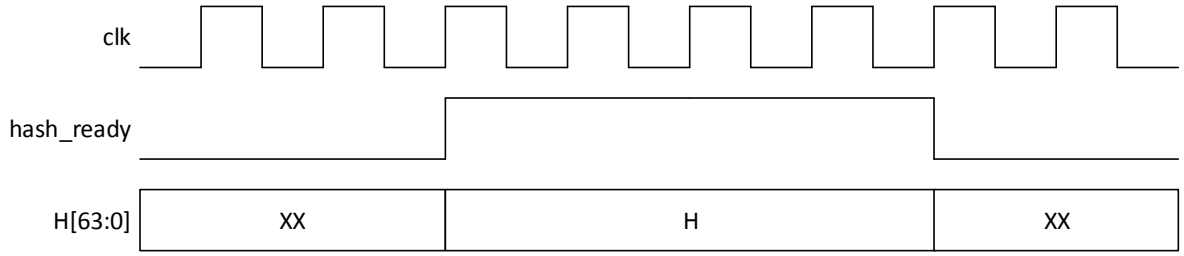
- assuming the ith byte $C[i] = C[i][7:0]$
- then $C_6[i] = \{C[i][7] \oplus C[i][1], C[i][3], C[i][2], C[i][5] \oplus C[i][0], C[i][4], C[i][6]\}$

Additional design specifications

- The stream cipher shall have an asynchronous active-low reset port;
- The input message byte M can be any 8-bit ASCII character;
- The stream cipher shall feature an input port which has to be asserted when providing the input message byte M (*M_valid* port): 1'b1, when input character is valid and stable, 1'b0, otherwise; the following waveform is expected at input interface of hash module



- The stream cipher shall feature an output port which is asserted when the generated output digest (or hash value) is available at the corresponding output port (*hash_ready* port): 1'b1, when output digest is valid and stable, 1'b0, otherwise; this flag shall be kept to logic 1 until a new message digest computation is performed; the following waveform is expected at the output interface of hash module



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 "01" bits and centrally the "1101" bits 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

- Design logic resources to perform the assignment of all the variables $H[i]$ in the same clock cycle.
- Maybe it could be easier to instantiate some resources (registers) dedicated to the computation of variables $H[i]$ for the main algorithm operation (i.e. when computation of $H[i]$ involves M and M_6), and some other resources (registers) dedicated to the computation of variables $H[i]$ for the last algorithm operation (i.e. when computation of $H[i]$ involves $C[i]$ and $C_6[i]$): such logic resources (registers) could be named, for instance, $H_{main}[i]$ and $H_{last}[i]$, begin $H_{last}[i]$ function of $C_6[i]$ and $H_{main}[i]$.
  Suggestion for project reports: which are the main differences, in terms of critical path and consumption of resources, between the implementation approach proposed above and an implementation approach in which the same resources (registers) $H[i]$ are used (to store) for both results of main algorithm operation and last algorithm operation?
- Logic resources (and maybe dedicated ports) are required to signal the beginning and the end of a message (by bytes): this is required to initialize the variables $H[i]$ and to trigger the last operation of the hash algorithm (i.e. the usage of C, instead of M).
- Develop testbench with messages of different length and check that the same message generates the same hash value.