Implementation and evaluation of SHA-3 candidates on FPGA

Extended abstract

Wim Ramakers
Hans Narinx

- Please alway's refer to the original document in Dutch:
  W. Ramakers, H. Narinx, "Implementatie en evaluatie van SHA-3-kandidaten op FPGA," M.S. thesis, Katholieke Hogeschool Limburg, Diepenbeek, België, 2010.
- The original thesis and VHDL code can be found online at:
  https://doks.khlim.be/do/lang?lang=en *(or e-mail wim_ramakers(at)telenet.be)*

Contents

Abstract

Due to the strong improvement of supercomputers and crypto analysis the safety of digital data becomes an issue. In the current hash functions, which are used to encrypt passwords and place digital signatures, several weaknesses are found. This means that there is no guarantee that the data we send and store on our computers is safe. The NIST has started a worldwide SHA-3 competition to find a new hash standard that is safe under current and future conditions in the digital world. This masters thesis wants to implement several new hash functions on FPGA, and evaluate and optimize them with regards to speed and area.

To simulate the implementations, software is used to verify that the implementation works correctly (Modelsim) on the one hand, and on the other hand to synthesize the VHDL code (Xilinx ISE). The synthesized code can be downloaded to an FPGA to test the implementation with real hardware.

The simulations, by software, give us a clear view of the differences between the new hash functions with regards to the speed and the use of area. They also clearly show us the impact of the optimisations, which were done to improve speed and/or reduce the size of area.

Introduction

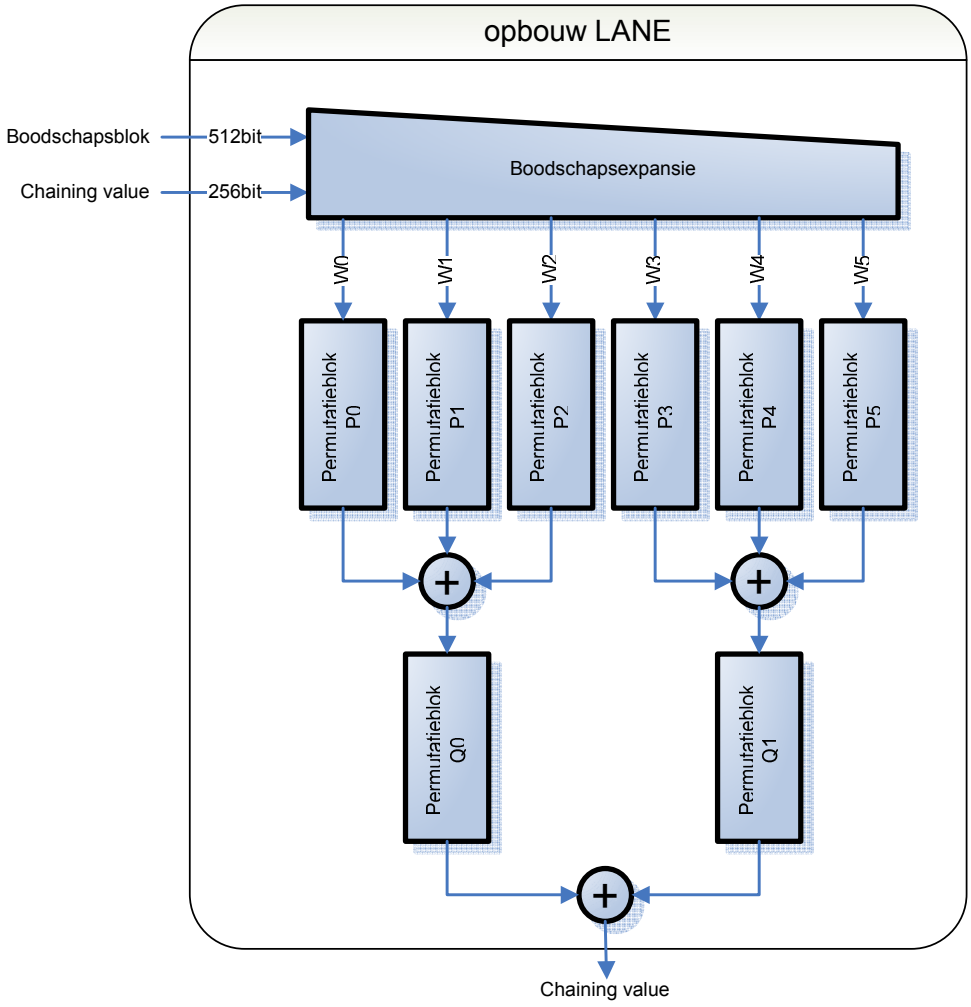
All results are for the compression functions only. For each hash function is also a fully autonomous implementation with data buffer available. All implementations are for 256 bit hash values, no salts.

Only the most important figures and most needed explanation are in this document. For more information see the original file in Dutch.

The aim of our thesis is implement some SHA-3 candidates in VHDL optimized for little use of area without losing to much speed/troughput.

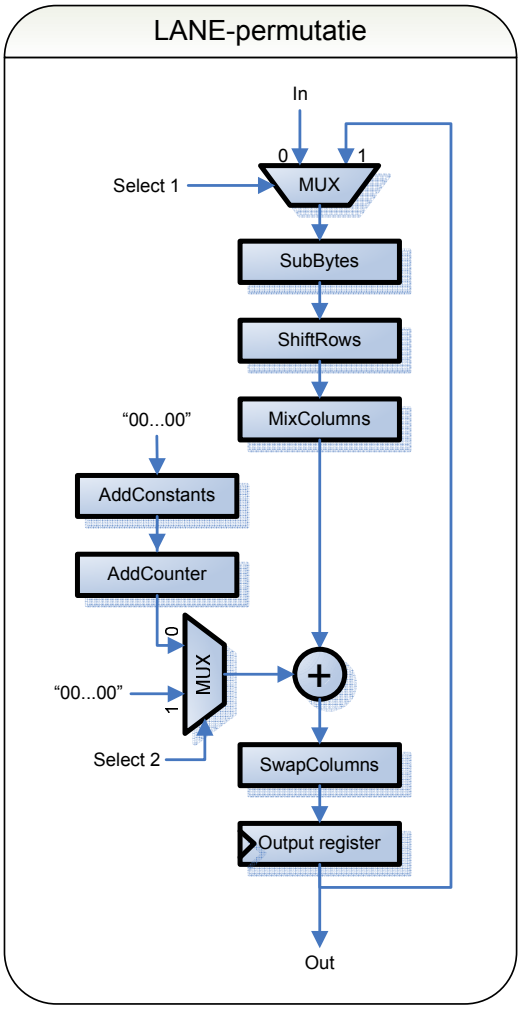
1 LANE

The LANE compression function is build like the figure below:

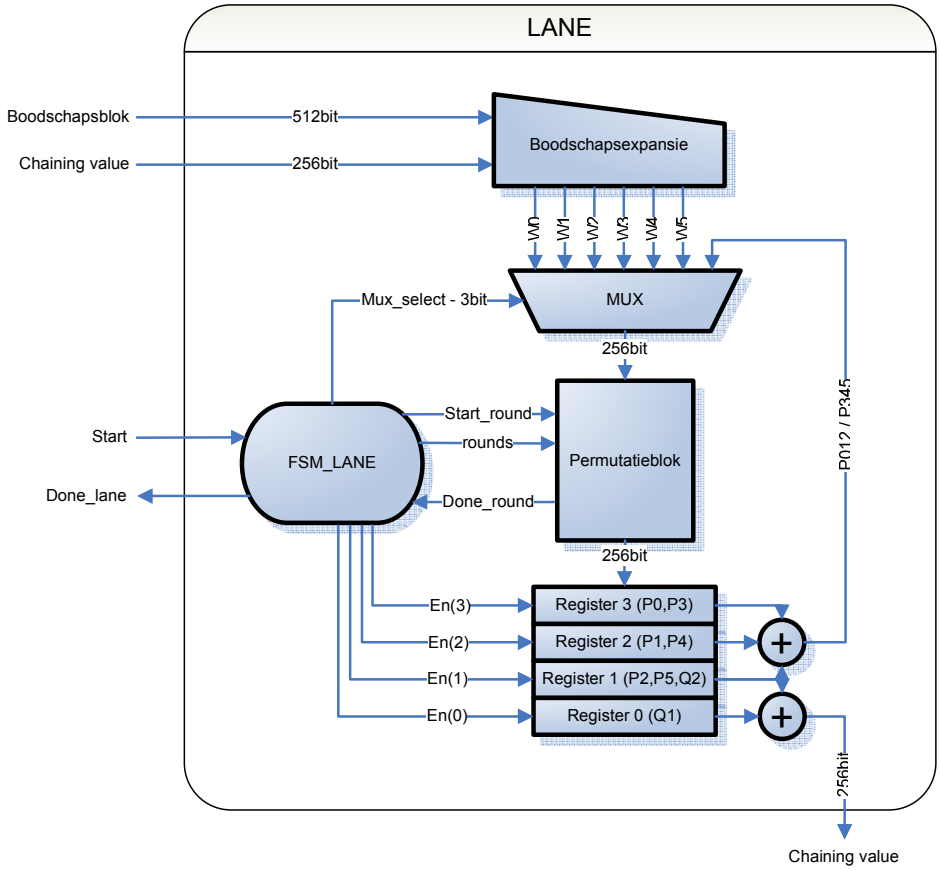


The permutation blocks

In de new implementation of LANE, the transformations are not all in one line. We can shorten the critical path by adding constants and counter trough EXOR's. The only difference between the P and Q permutation is the number of rounds. We use a general permutation block with a selection of number of rounds.



To make the implementation smaller, we only use one permutation block and save the output in registers. If we make a good choice in the order of execution of each permutation, we only need four registers.

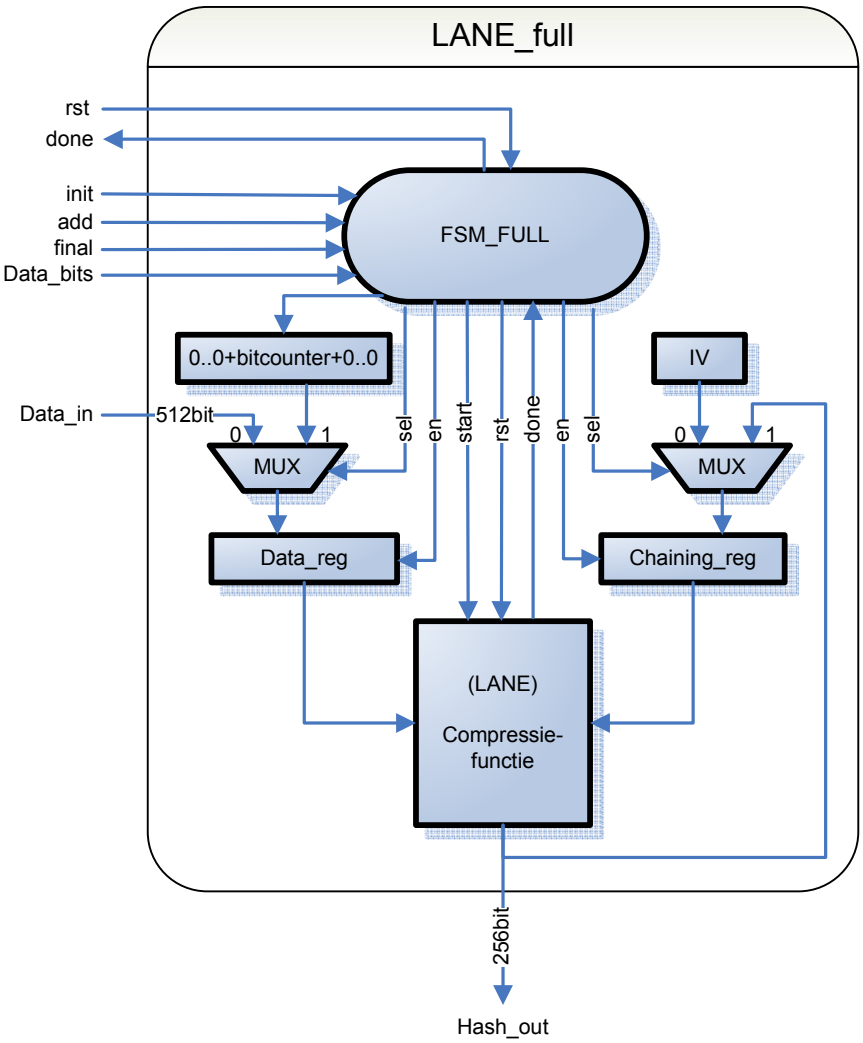


The results are shown in the table below.

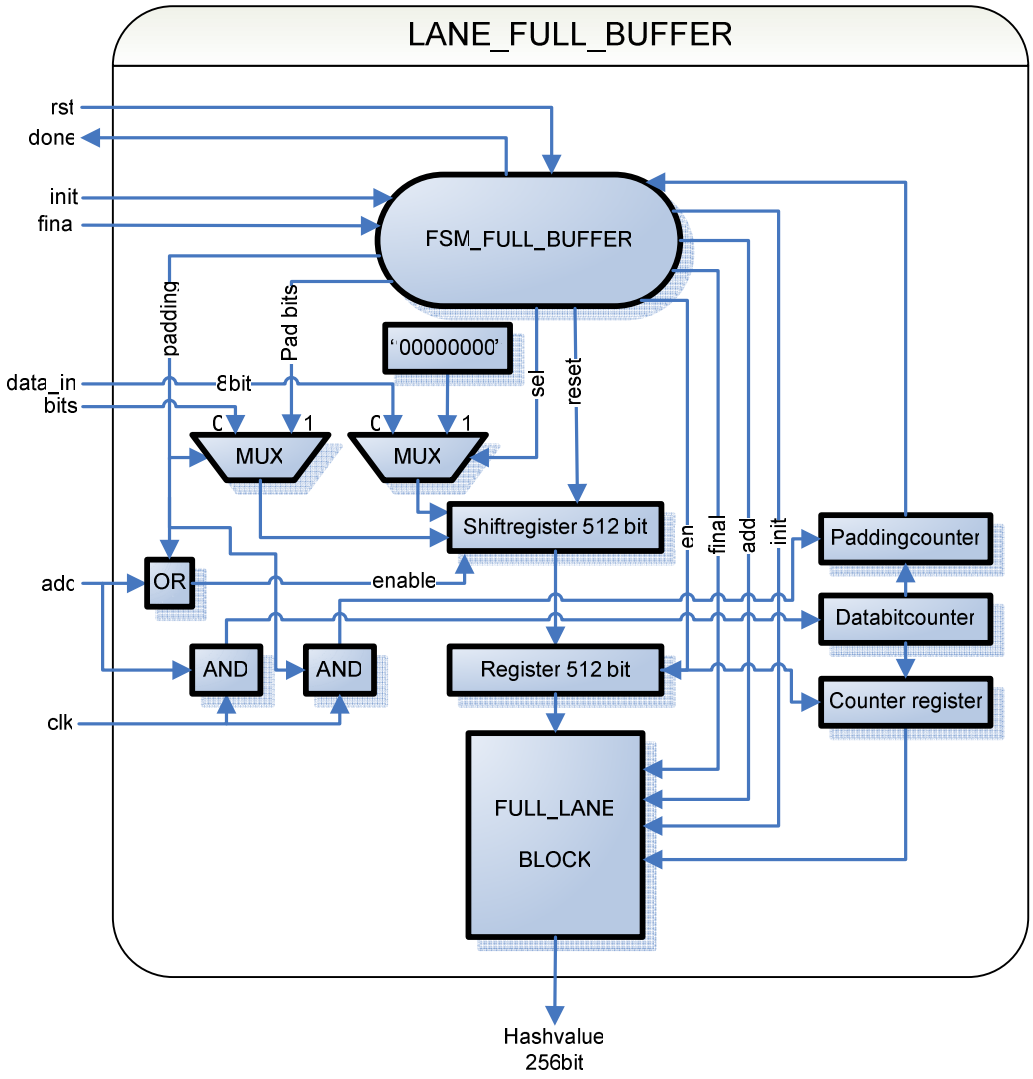
Xilinx Virtex-5 FPGA	**LANE_old [2]**	**LANE_new**	**LANE [11]**
Number of slices	*8228*	*3389*	*3442*
Number of Slice registers	*2111*	*1307*	*/*
Number of slice LUT's	*16600*	*4078*	*/*
Minimal period	*4.067 ns*	*5.622 ns*	*7.519 ns*
Estimated max clock frequency	*245 MHz*	*177 MHz*	*133 MHz*
Number of clock cycles per compression	*15*	*61*	*49*
Time for 1 compression	*61 ns*	*343 ns*	*369 ns*
Throughput	*8.36 Gbps*	*1.48 Gbps*	*1.38 Gbps*
Throughput /Area	*1.01 Mbps/slice*	*0.43 Mbps/slice*	*0.4 Mbps/slice*

Fully autonomous buffer

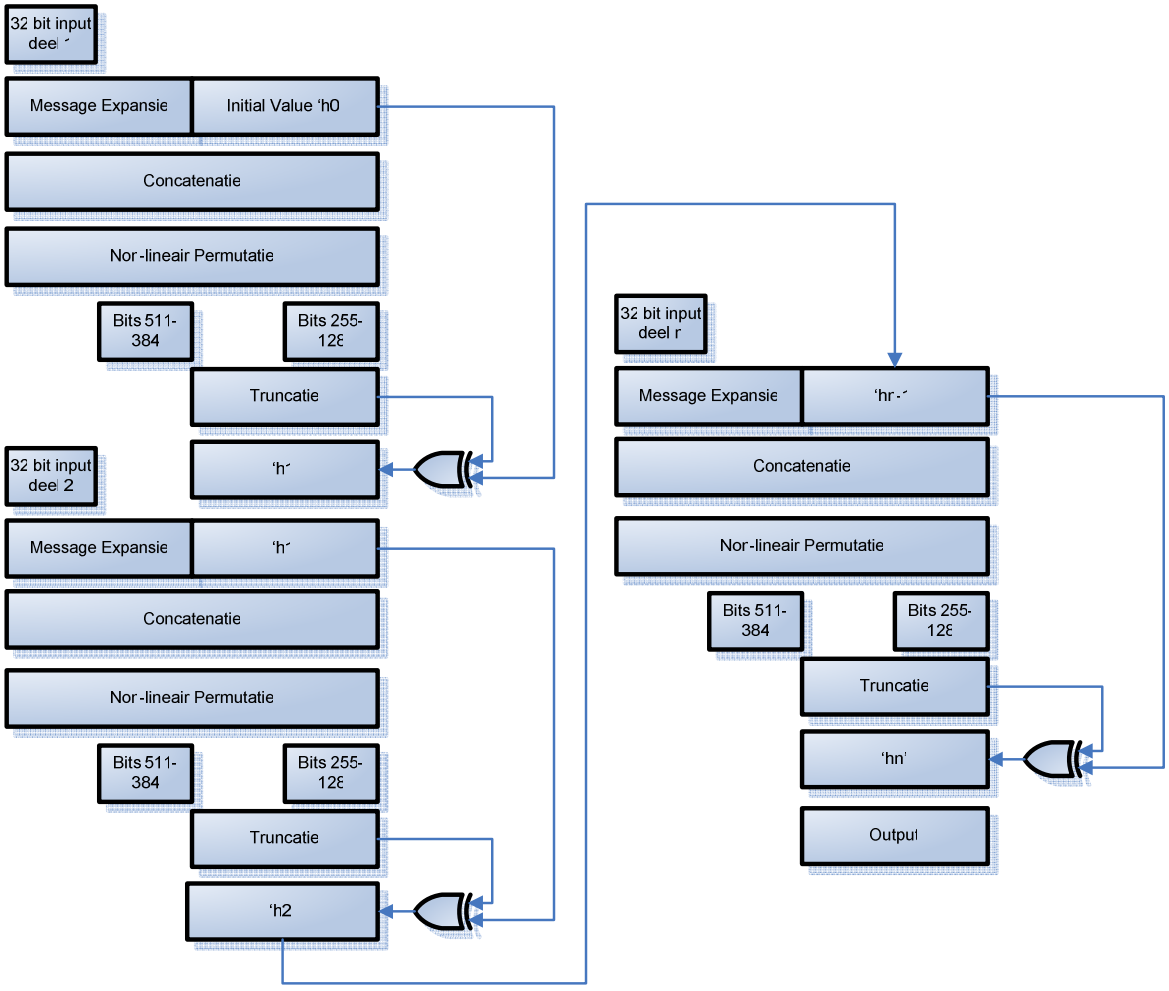
If we provide the last data already padded with zeros, this block can handle the full hash process.

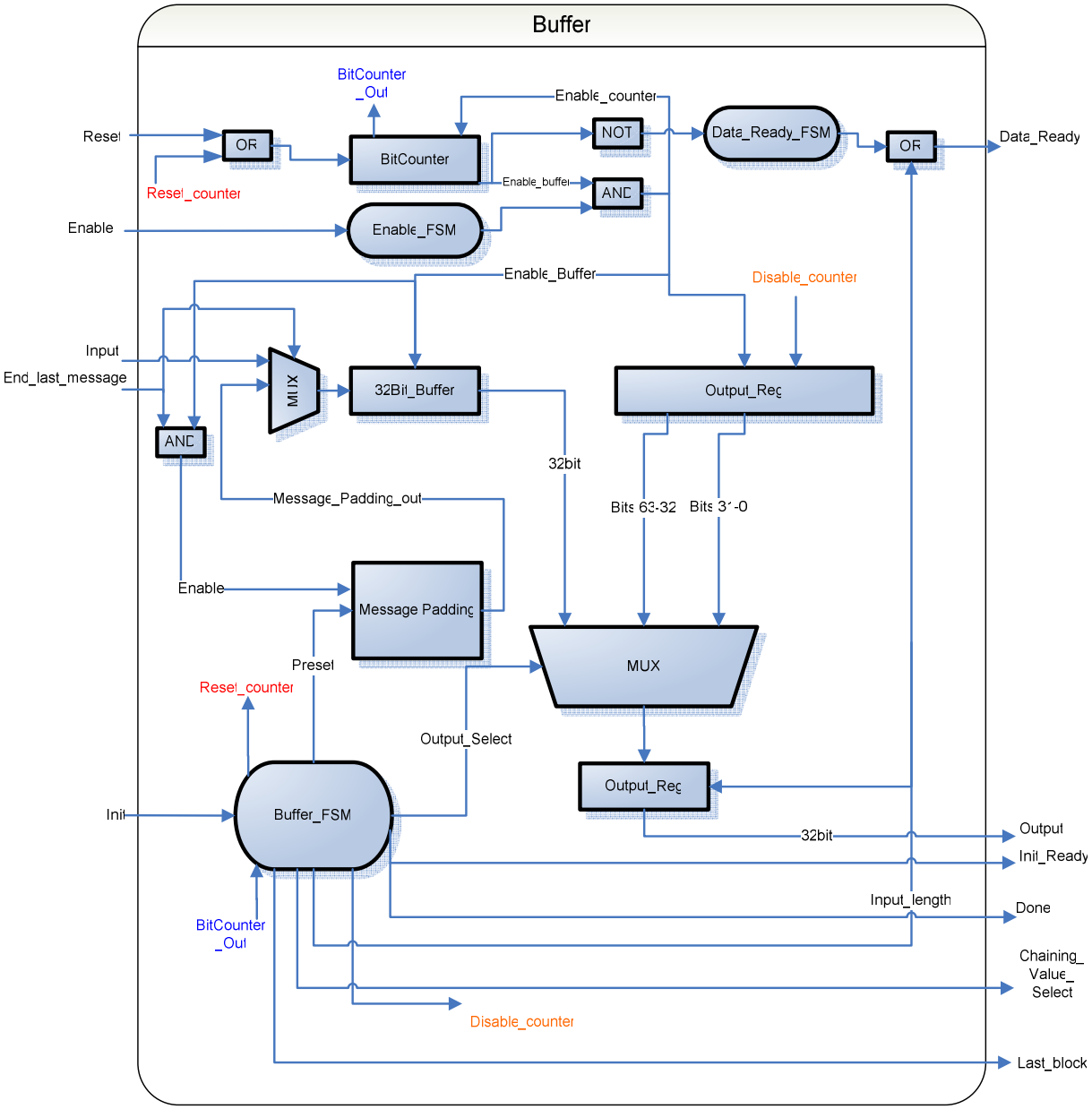


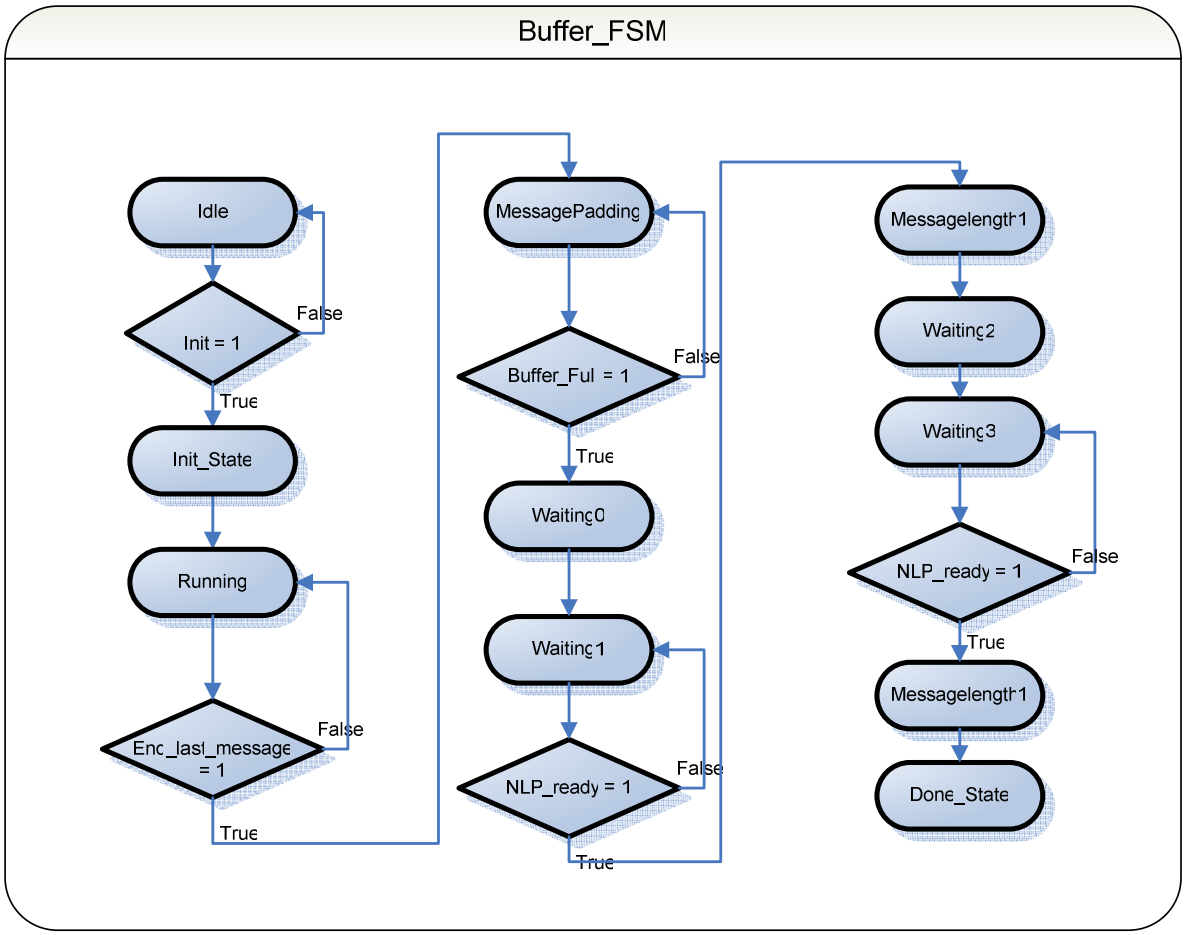
We also implemented a buffer which hashes automatical on buffer overflows and does message padding autonomous. It takes byte or bit input.

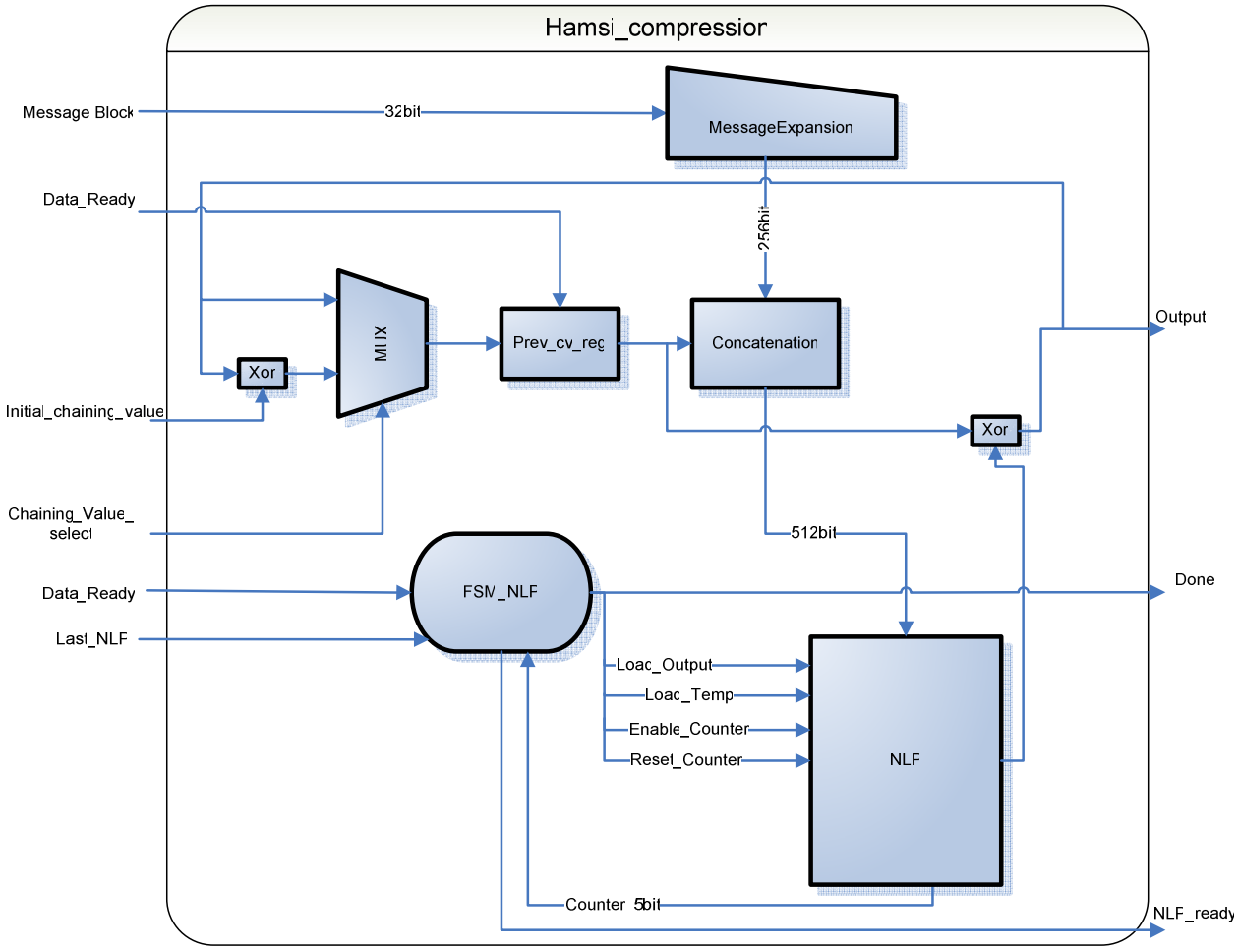

LANE_FULL_BUFFER
rst
done
init
fina
FSM_FULL_BUFFER
padding
Pad bits
'00000000'
data_in
bits
8bit
sel
reset
0
1
MUX
MUX
Shiftregister 512 bit
en
final
add
init
Paddingcounter
adc
OR
enable
Databitcounter
AND
AND
Register 512 bit
Counter register
clk
FULL_LANE
BLOCK
Hashvalue
256bit

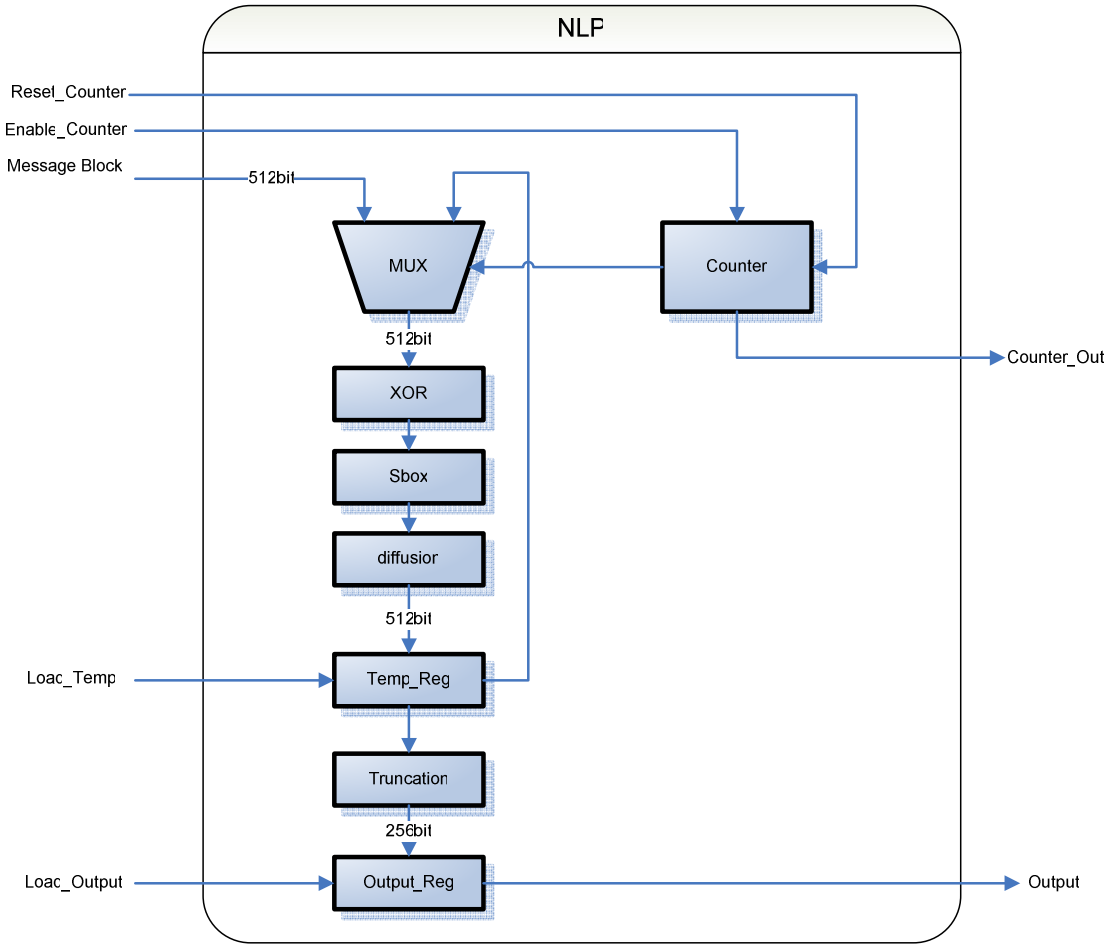
2 Hamsi

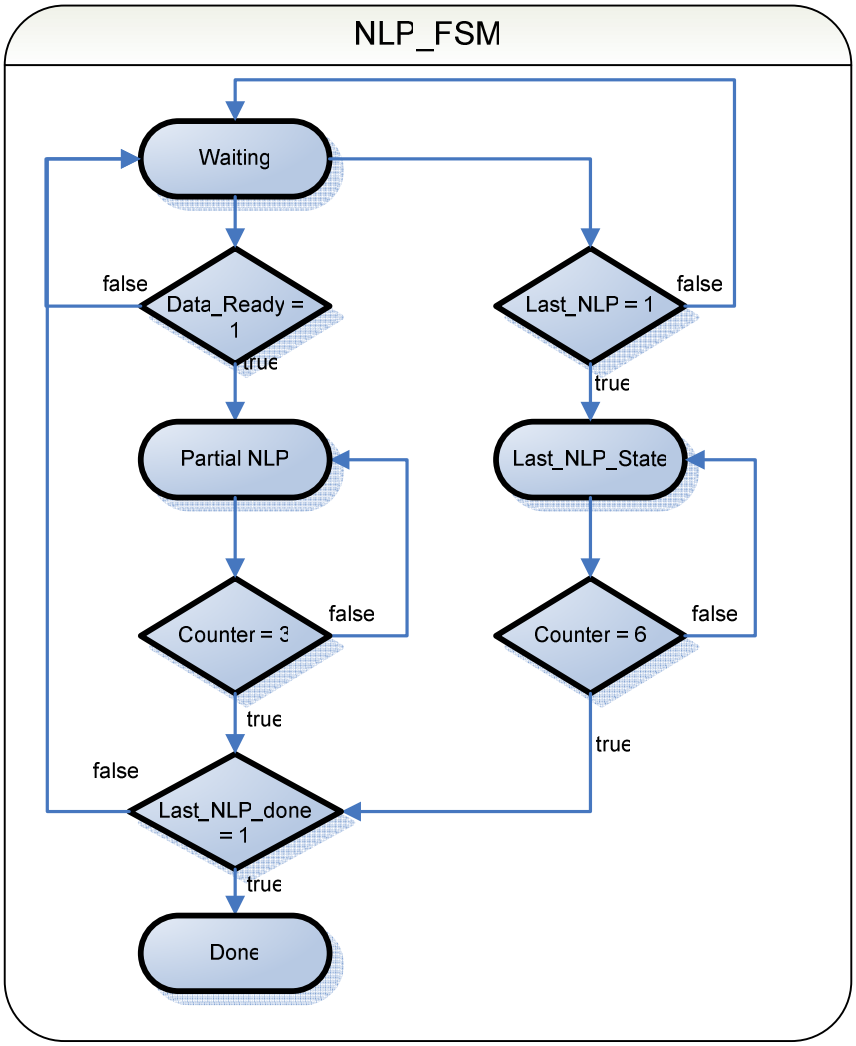

32 bit input
Message Expansie
Initial Value 'h0
Concatenatie
Nor-lineair Permutatie
Bits 511-384
Bits 255-128
Truncatie
'h
32 bit input
deel 2
Message Expansie
'h
Concatenatie
Nor-lineair Permutatie
Bits 511-384
Bits 255-128
Truncatie
'h2
32 bit input
deel r
Message Expansie
'hr
Concatenatie
Nor-lineair Permutatie
Bits 511-384
Bits 255-128
Truncatie
'hn
Output


Buffer
BitCounter
_Out
Enable_counter
Reset
OR
BitCounter
NOT
Data_Ready_FSM
OR
Data_Ready
Reset_counter
Enable_buffer
AND
Enable
Enable_FSM
Enable_Buffer
Disable_counter
Input
End_last_message
MUX
32Bit_Buffer
Output_Reg
AND
32bit
Message_Padding_out
Bits 63-32
Bits 31-0
Enable
Message Padding
Preset
MUX
Reset_counter
Output_Select
Output_Reg
Init
Buffer_FSM
32bit
Output
Init_Ready
Input_length
Done
BitCounter
_Out
Chaining_
Value_
Select
Disable_counter
Last_block

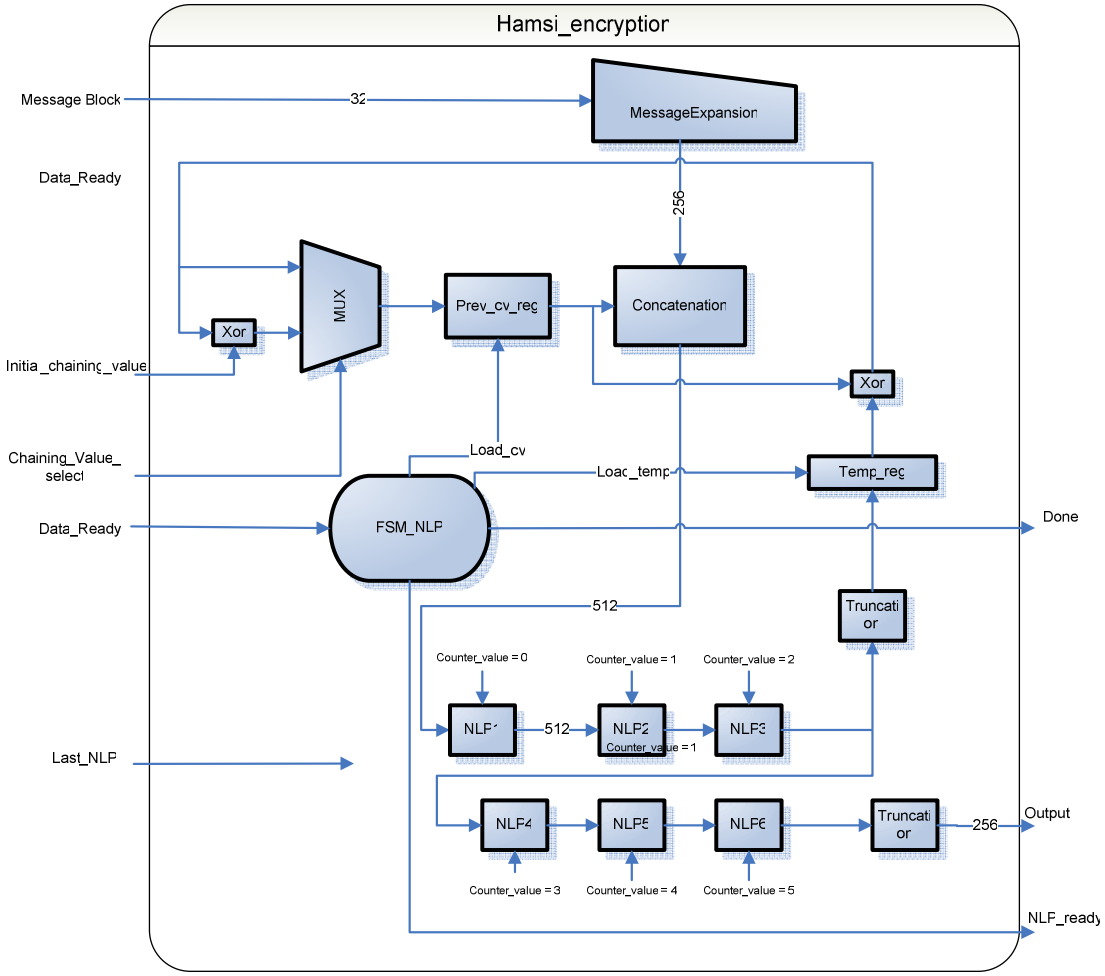

Buffer_FSM
Idle
Init = 1
False
True
Init_State
Running
Enc_last_message = 1
False
True
MessagePadding
Buffer_Ful = 1
False
True
Waiting0
Waiting1
NLP_ready = 1
False
True
Messagelength1
Waiting2
Waiting3
NLP_ready = 1
False
True
Messagelength1
Done_State


Hamsi_compression
Message Block
32bit
MessageExpansion
Data_Ready
256bit
Output
MUX
Prev_cv_reg
Concatenation
Xor
Initial_chaining_value
Xor
Chaining_Value_
select
512bit
Done
Data_Ready
FSM_NLF
Last_NLF
Loac_Output
Loac_Temp
Enable_Counter
Reset_Counter
NLF
Counter 5bit
NLF_ready


NLP
Reset_Counter
Enable_Counter
Message Block
512bit
MUX
Counter
512bit
Counter_Out
XOR
Sbox
diffusior
512bit
Loac_Temp
Temp_Reg
Truncation
256bit
Loac_Output
Output_Reg
Output


NLP_FSM
Waiting
false
Data_Ready = 1
true
Last_NLP = 1
false
true
Partial NLP
Last_NLP_State
Counter = 3
false
Counter = 6
false
true
true
false
Last_NLP_done = 1
true
Done

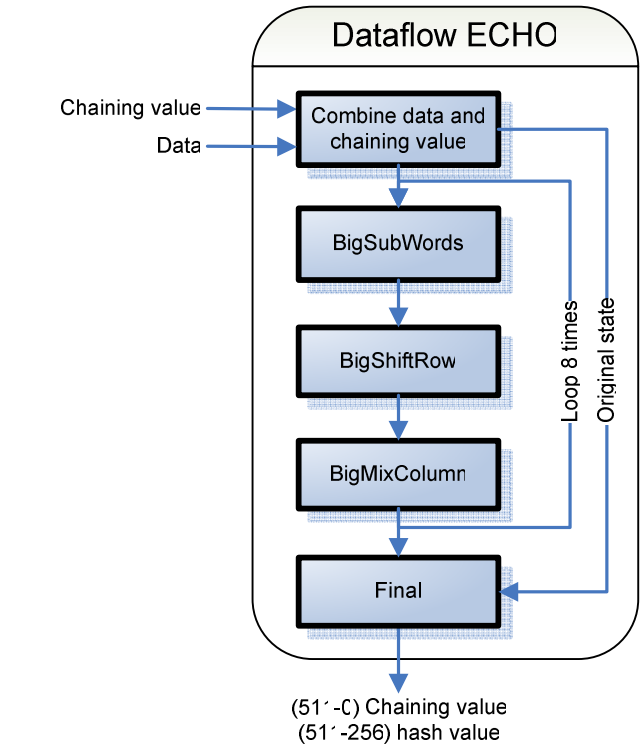
.

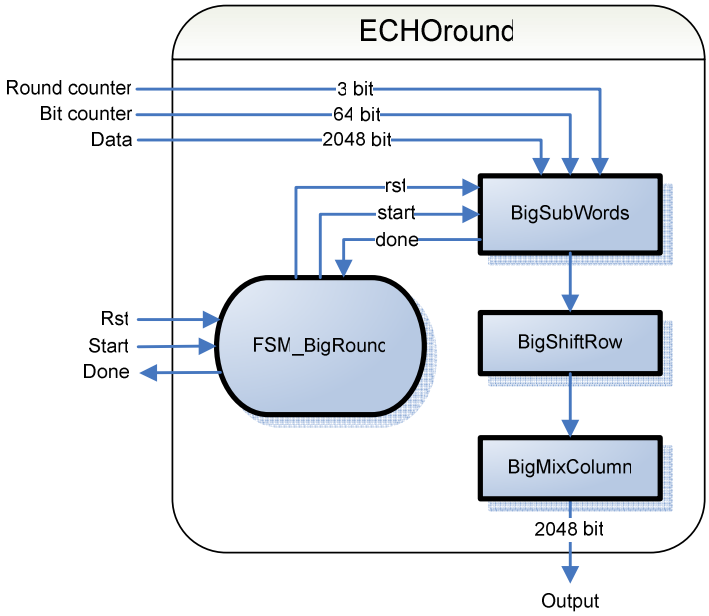

Hamsi_encryptior
Message Block
32
MessageExpansion
Data_Ready
256
MUX
Xor
Prev_cv_reg
Concatenation
Initia_chaining_value
Xor
Load_cv
Chaining_Value_
select
Loac_temp
Temp_reg
Data_Ready
FSM_NLP
Done
512
Truncati
or
Counter_value = 0
Counter_value = 1
Counter_value = 2
NLP1
512
NLP2
Counter_value = 1
NLP3
Last_NLP
NLP4
NLP5
NLP6
Truncati
or
256
Output
Counter_value = 3
Counter_value = 4
Counter_value = 5
NLP_ready

Xilinx Virtex-5 FPGA	**Hamsi-256 fast**	**Hamsi-256 small**	**Hamsi[13]**
Number of slices	*4,664*	*2113*	*733*
Number of Slice registers	*514*	*781*	*/*
Number of slice LUT's	*7216*	*3004*	*/*
Minimal period	*4.826*	*3.242 ns*	*3.484 ns*
Estimated max clock frequency	*207 MHz*	*308 MHz*	*287 MHz*
Number of clock cycles per compression	*1*	*5*	*7*
Time for 1 compression	*24 ns*	*16 ns*	*24 ns*
Throughput	*6.62 Gbps*	*1.97 Gbps*	*1.48 Gbps*
Throughput /Area	*1.42 Mbps/slice*	*0.93 Mbps/slice*	*2 Mbps/slice*

3 ECHO

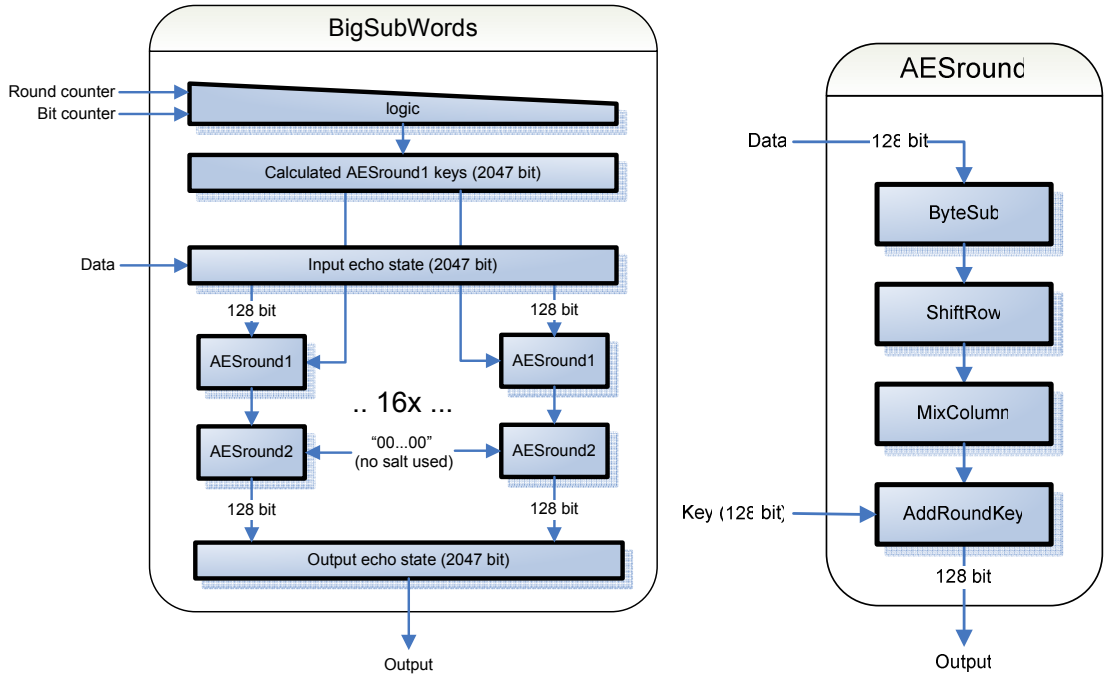
In the figure below we see the dataflow of ECHO.

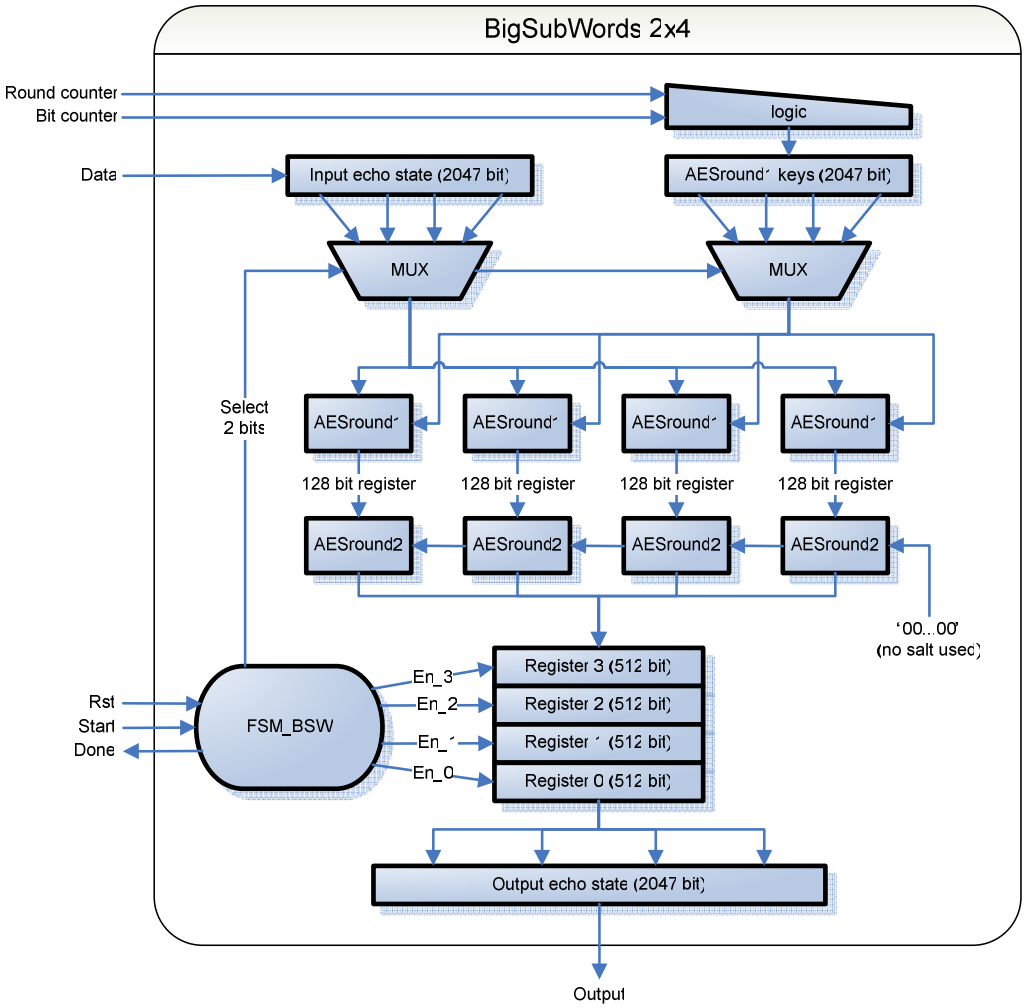




BigSubWords is the only block that can be optimized simple without losing to much speed.

BigSubWords



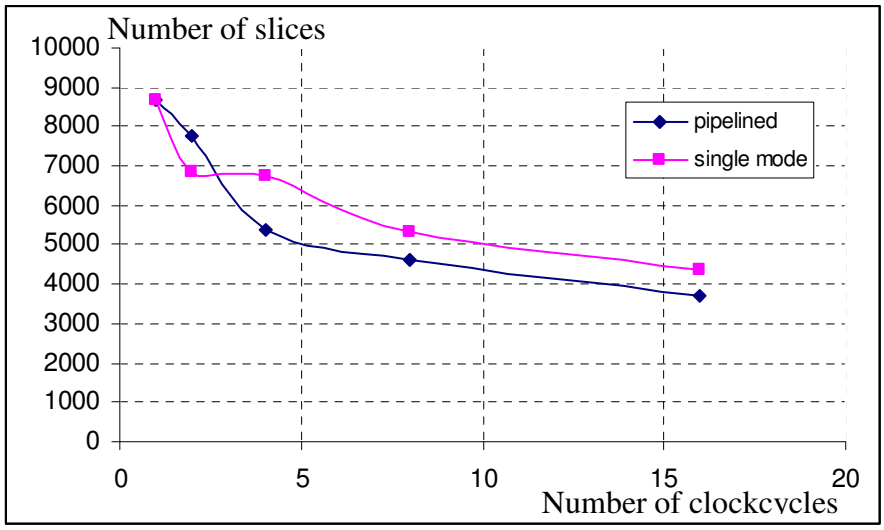


Overview pipelined implementations

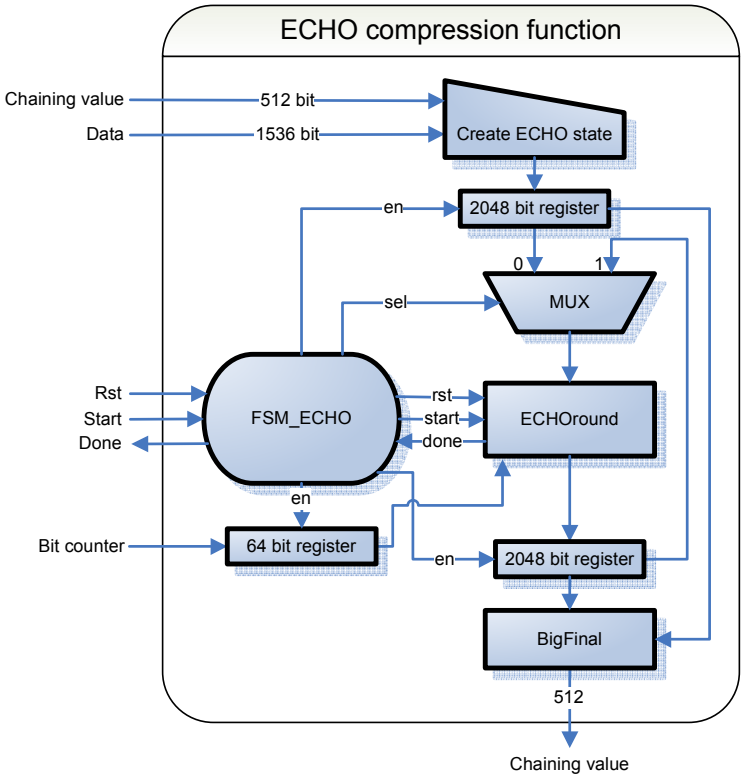
Virtex 5 XC5VLX155T	**ref**	**2 x 8**	**2 x 4**	**2 x 2**	**2 x 1**
Maximal frequency	*95MHz*	*266MHz*	*247MHz*	*203MHz*	*224MHz*
Number of clock cycles	*1*	*3*	*5*	*9*	*17*
Number of Slices	*8659*	*7763*	*5372*	*4635*	*3702*

Overview single mode implementations

Virtex 5 XC5VLX155T	**ref**	**1 x 16**	**1 x 8**	**1 x 4**	**1 x 2**
Maximal frequency	*95MHz*	*266MHz*	*261MHz*	*227MHz*	*200MHz*
Number of clock cycles	*1*	*2*	*4*	*8*	*16*
Number of Slices	*8659*	*6864*	*6743*	*5353*	*4357*

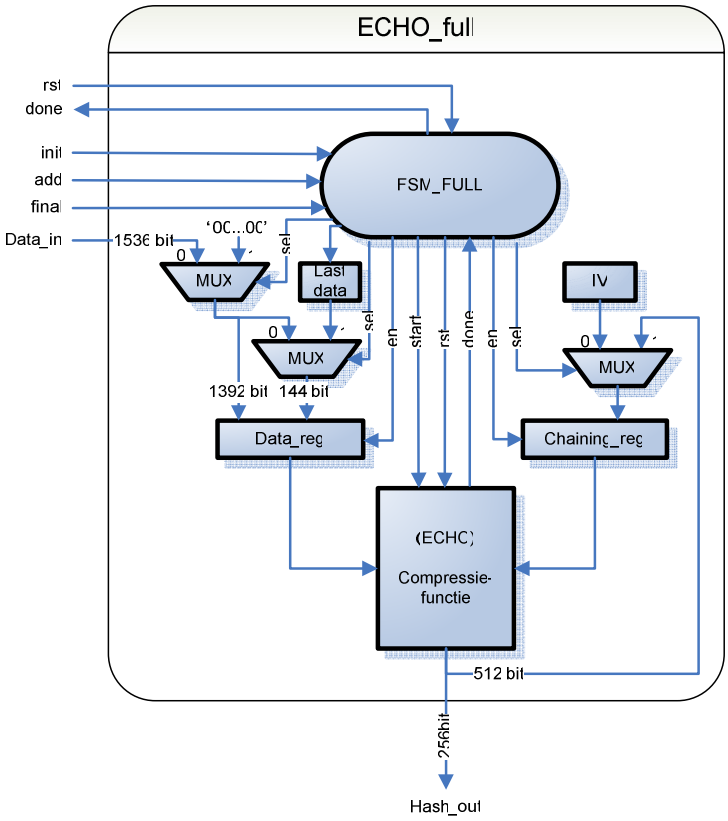


For the optimized version in the table we chose the 2x4 implementation.

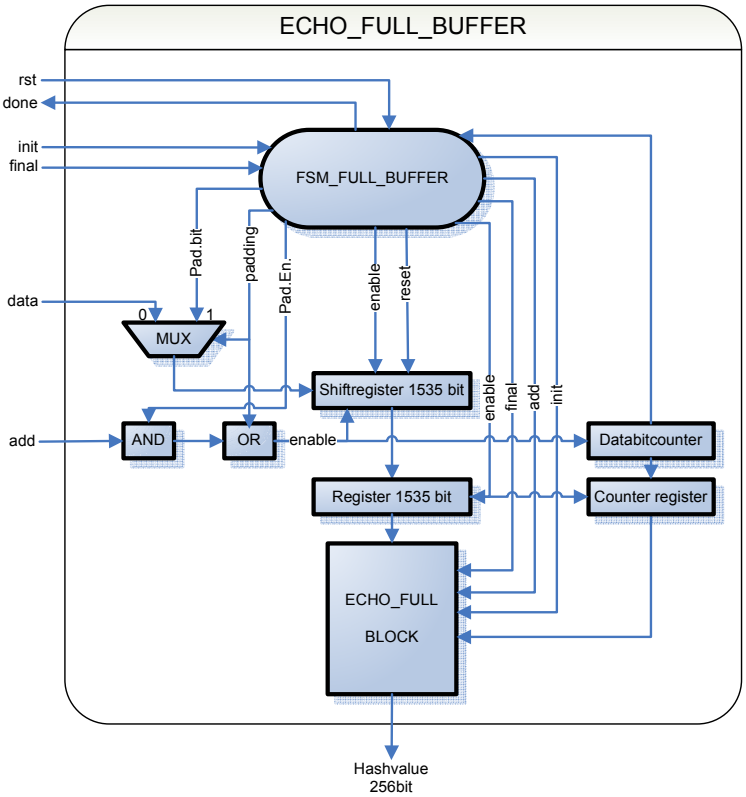


Xilinx Virtex-5 FPGA	**ECHO standard**	**ECHO optimized**
Number of slices	*15006*	*12061*
Number of Slice registers	*4105*	*8800*
Number of slice LUT's	*29330*	*14407*
Minimal period	*7.154 ns*	*5.333 ns*
Estimated max clock frequency	*139 MHz*	*187 MHz*
Number of clock cycles per compression	*9*	*81*
Time for 1 compression	*64 ns*	*432 ns*
Throughput	*23.86 Gbps*	*3.56 Gbps*
Throughput /Area	*1.59 Mbps/slice*	*0.30 Mbps/slice*

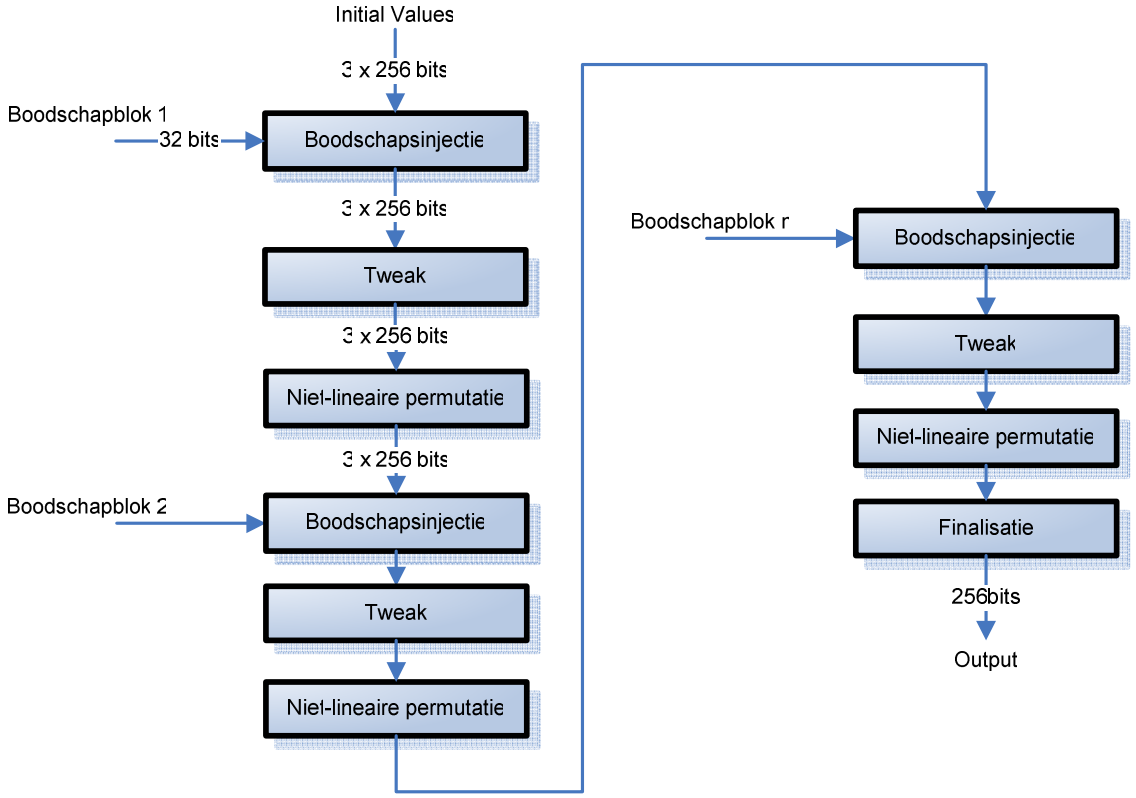
Fully autonomous buffer

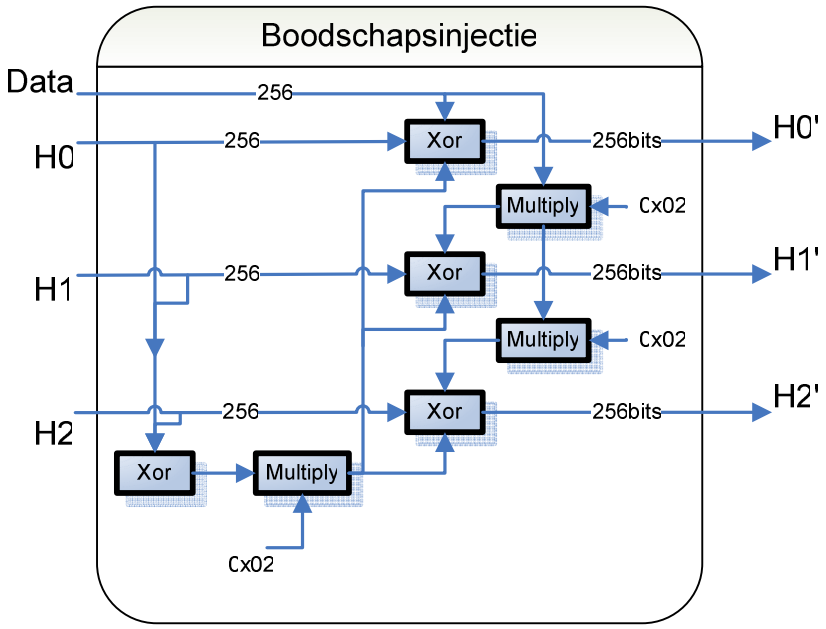


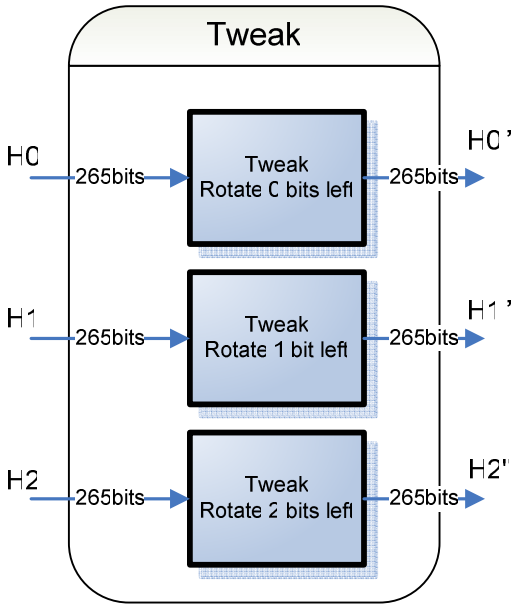
When you pad the message yourselve with a one, this block can handle the full hash procedure. Also here a databuffer was made which does everything autonomous (See picture below).

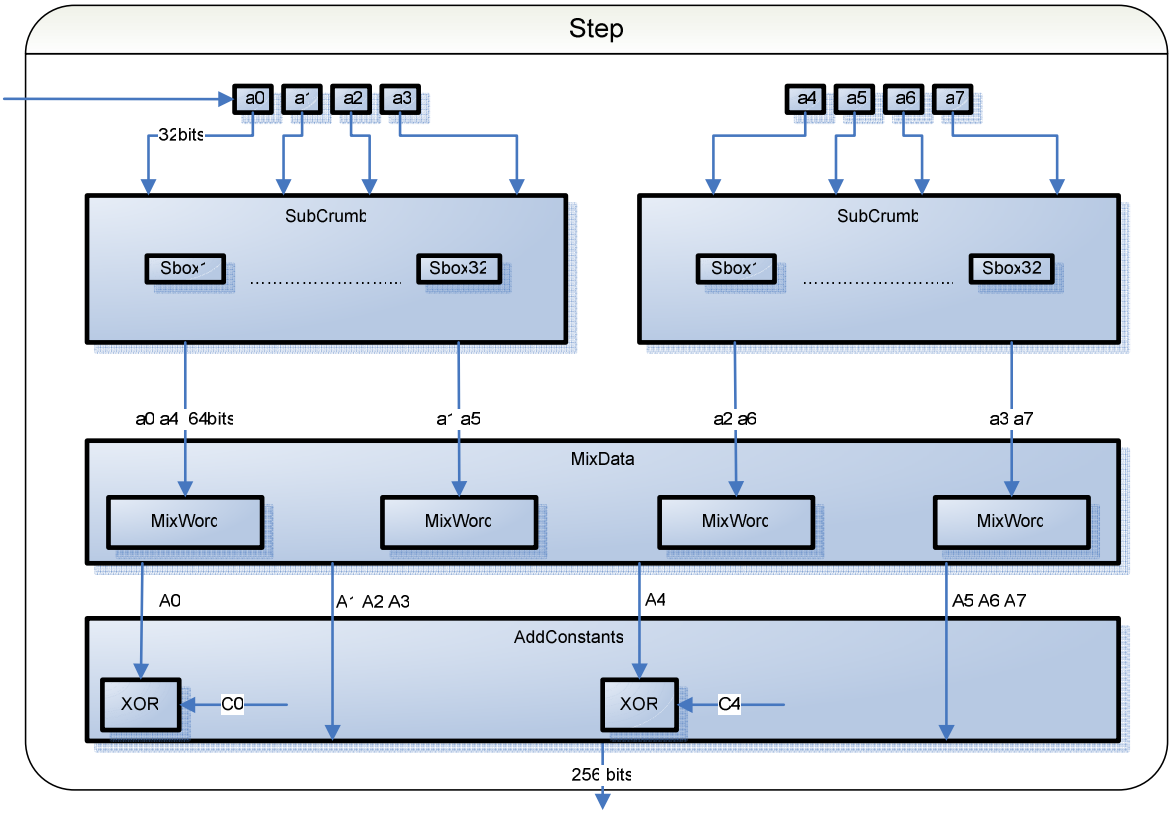


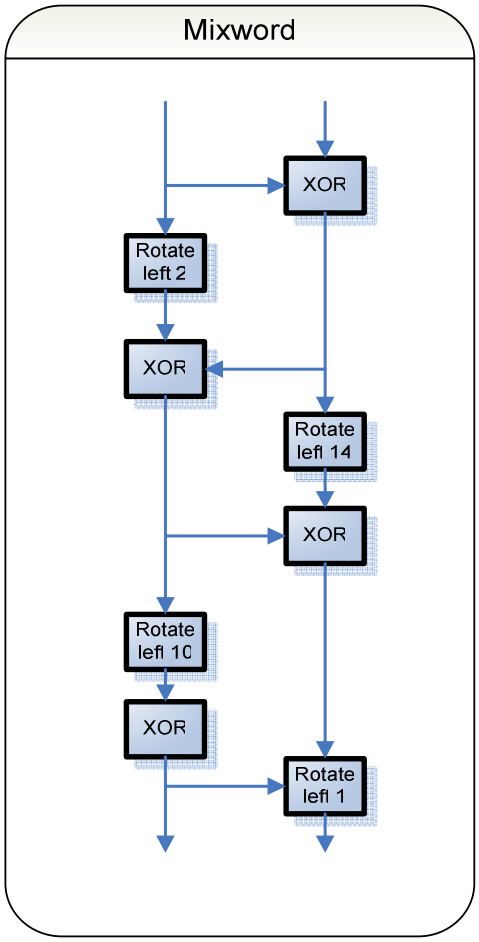
4 Luffa

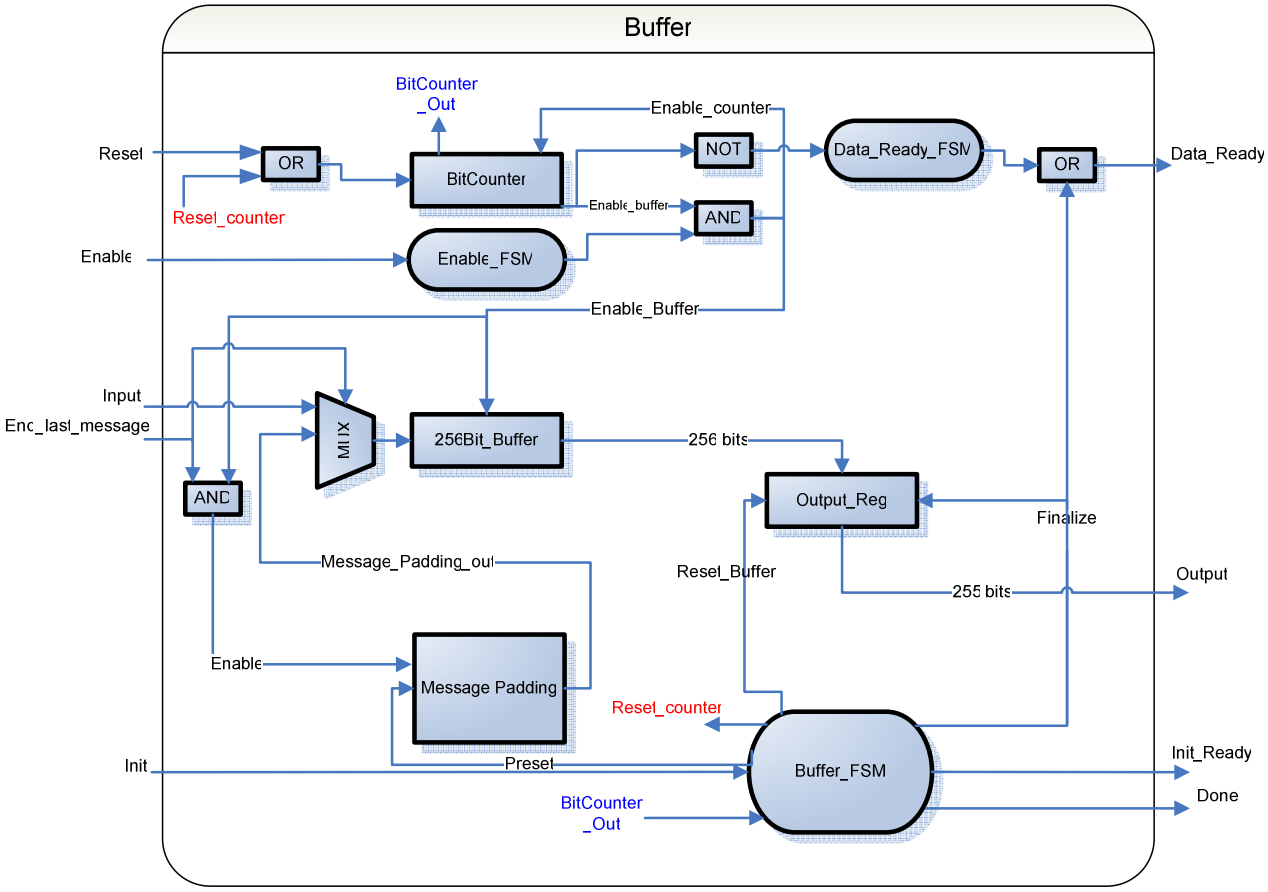

Initial Values
3 x 256 bits
Boodschapblok 1
32 bits
Boodschapsinjectie
3 x 256 bits
Tweak
3 x 256 bits
Niet-lineaire permutatie
3 x 256 bits
Boodschapblok 2
Boodschapsinjectie
Tweak
Niet-lineaire permutatie
Boodschapblok r
Boodschapsinjectie
Tweak
Niet-lineaire permutatie
Finalisatie
256bits
Output

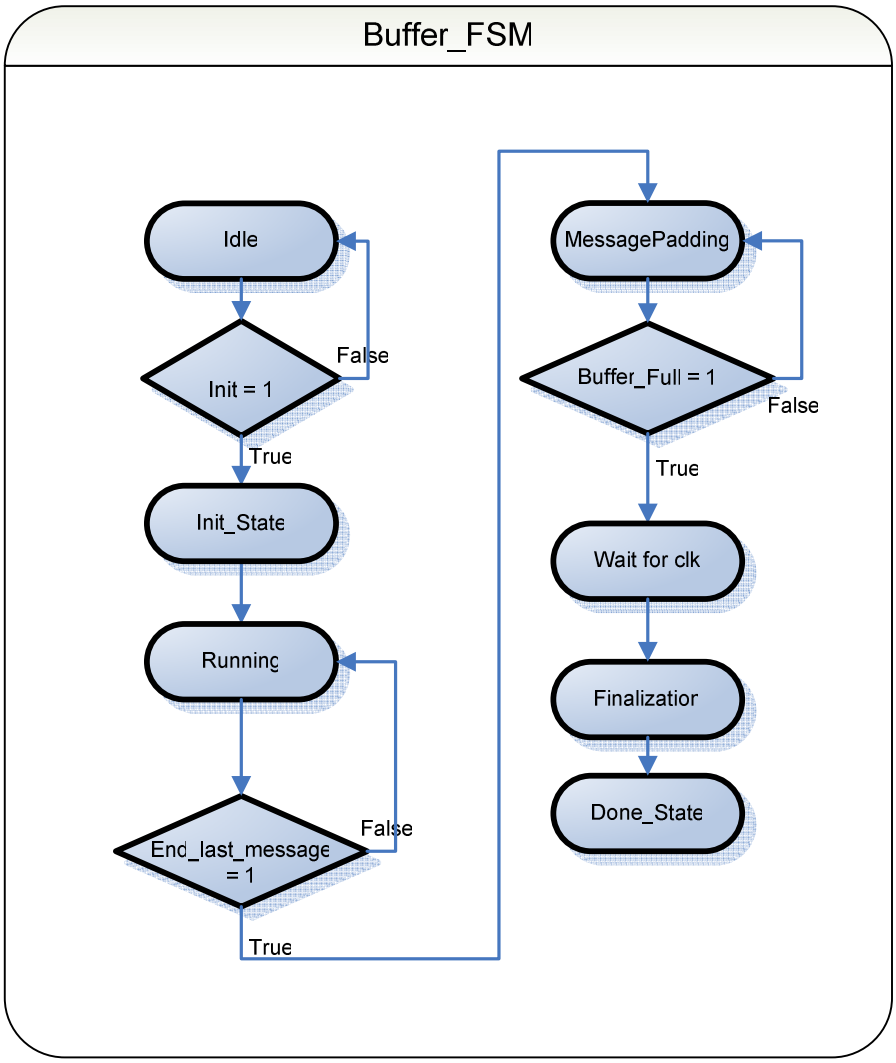

Boodschapsinjectie
Data
256
H0
256
Xor
256bits
H0'
Multiply
Cx02
H1
256
Xor
256bits
H1'
Multiply
Cx02
H2
256
Xor
256bits
H2'
Xor
Multiply
Cx02


Tweak
H0
265bits
Tweak
Rotate 0 bits left
265bits
H0'
H1
265bits
Tweak
Rotate 1 bit left
265bits
H1'
H2
265bits
Tweak
Rotate 2 bits left
265bits
H2'

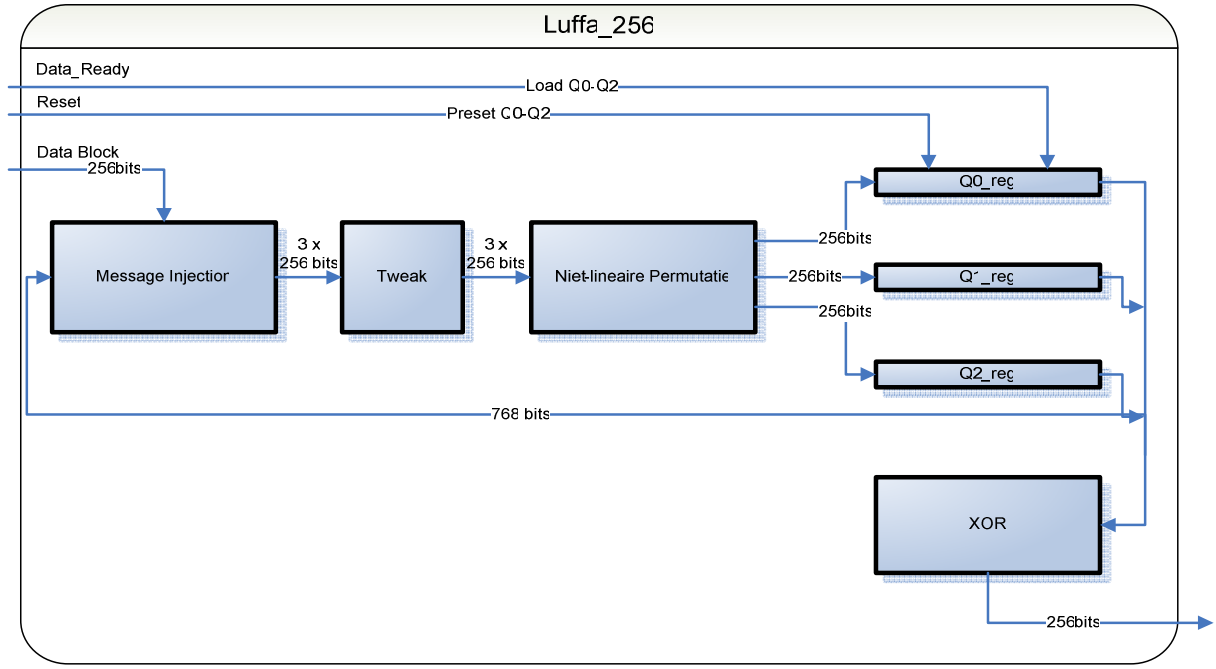

Step
a0
a1
a2
a3
32bits
SubCrumb
Sbox1
Sbox32
a4
a5
a6
a7
SubCrumb
Sbox1
Sbox32
a0 a4 64bits
a1 a5
a2 a6
a3 a7
MixData
MixWord
MixWord
MixWord
MixWord
A0
A1 A2 A3
A4
A5 A6 A7
AddConstants
XOR
C0
XOR
C4
256 bits


Mixword
XOR
Rotate left 2
XOR
Rotate left 14
XOR
Rotate left 10
XOR
Rotate left 1

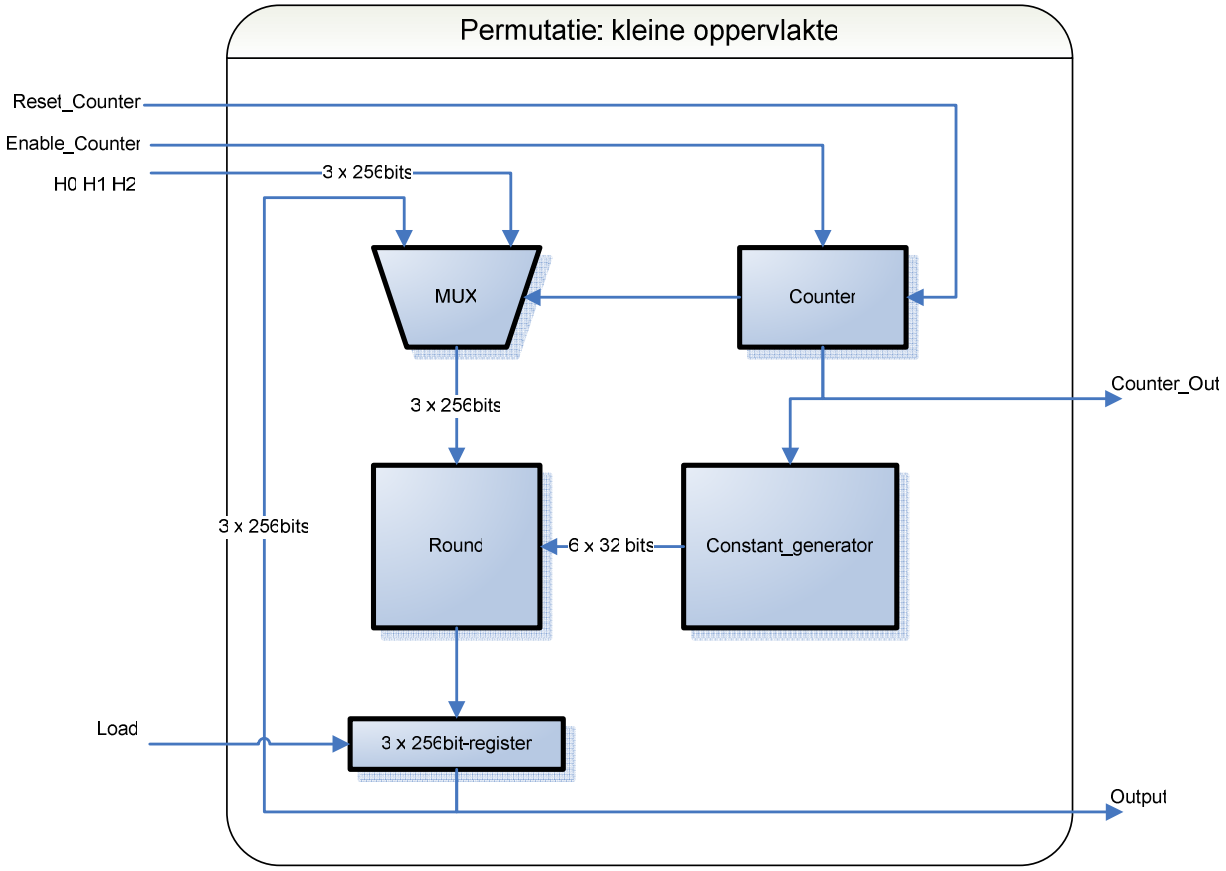

Buffer
BitCounter
_Out
Enable_counter
Reset
OR
BitCounter
NOT
Data_Ready_FSM
OR
Data_Ready
Reset_counter
Enable_buffer
AND
Enable
Enable_FSM
Enable_Buffer
Input
Enc_last_message
MUX
256Bit_Buffer
256 bits
AND
Output_Reg
Finalize
Message_Padding_out
Reset_Buffer
Output
255 bits
Enable
Message Padding
Reset_counter
Buffer_FSM
Init_Ready
Init
Preset
Done
BitCounter
_Out

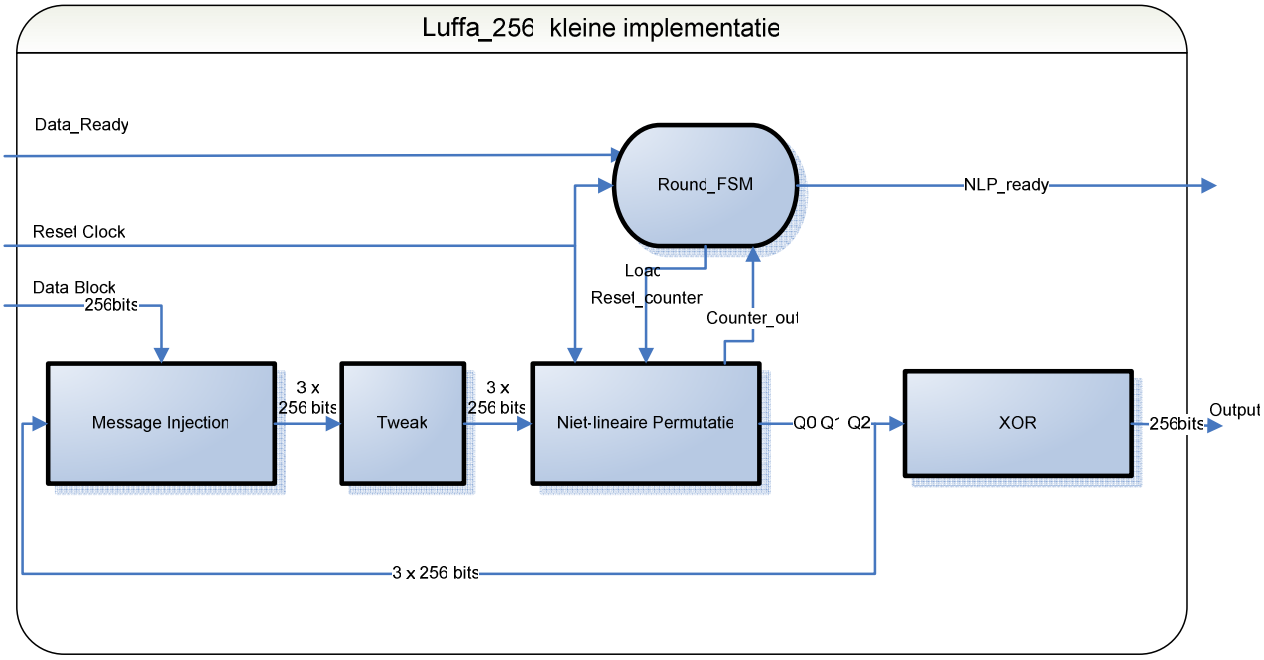


Speed optimized



Area optimized





Xilinx Virtex-5 FPGA	**Luffa_256 (Comb)**	**Luffa_256 (Seq)**
Number of slices	*9611*	*2303*
Number of Slice registers	*786*	*1165*
Number of slice LUT's	*18918*	*4490*
Minimal period	*20.739 ns*	*5.568 ns*
Estimated max clock frequency	*48.2 MHz*	*179 MHz*
Number of clock cycles per compression	*1*	*9*
Time for 1 compression	*20.739 ns*	*44.544 ns*
Throughput	*12.29 Gbps*	*5.09 Gbps*
Throughput /Area	*1.28 Mbps/slice*	*2.21 Mbps/slice*

3 Comparison

The smallest implementations for each hashfunction are in the table below.

Tabel 1: Vergelijking van de kleinste implementaties

Xilinx Virtex-5 FPGA	**LANE_nieuw**	**Hamsi [13]**	**ECHO Geoptimaliseerd**	**Luffa_ Seq**
Number of slices	*3389*	*733*	*12061*	*2303*
Number of Slice registers	*1307*	*/*	*8800*	*1165*
Number of slice LUT's	*4078*	*/*	*14407*	*4490*
Minimal period	*5.622 ns*	*3.484 ns*	*5.333 ns*	*5.568 ns*
Estimated max clock frequency	*177 MHz*	*287 MHz*	*187 MHz*	*179 MHz*
Number of clock cycles per compression	*61*	*7*	*81*	*9*
Time for 1 compression	*343 ns*	*24 ns*	*432 ns*	*44.544 ns*
Throughput	*1.48 Gbps*	*1.48 Gbps*	*3.56 Gbps*	*5.09 Gbps*
Throughput /Area	*0.43 Mbps/slice*	*2 Mbps/slice*	*0.30 Mbps/slice*	*2.21 Mbps/slice*

The fastest implementations for each hashfunction are in the table below.

Tabel 2: Vergelijking van de snelste implementaties

Xilinx Virtex-5 FPGA	**LANE_oud**	**Hamsi_fast**	**ECHO standaard**	**Luffa_comb**
Number of slices	*8228*	*4,664*	*15006*	*9611*
Number of Slice registers	*2111*	*514*	*4105*	*786*
Number of slice LUT's	*16600*	*7216*	*29330*	*18918*
Minimal period	*4.067 ns*	*4.826*	*7.154 ns*	*20.739 ns*
Estimated max clock frequency	*245 MHz*	*207 MHz*	*139 MHz*	*48.2 MHz*
Number of clock cycles per compression	*15*	*1*	*9*	*1*
Time for 1 compression	*61 ns*	*24 ns*	*64 ns*	*20.739 ns*
Throughput	*8.36 Gbps*	*6.62 Gbps*	*23.86 Gbps*	*12.29 Gbps*
Throughput /Area	*1.01 Mbps/slice*	*1.42 Mbps/slice*	*1.59 Mbps/slice*	*1.28 Mbps/slice*

Bibliography

[1] N. Mentens, “Secure and Efficient Coprocessor Design for Cryptographic Applications on FPGAs,” PhD thesis, Katholieke Universiteit Leuven, Leuven-Heverlee, België, 2007.

[2] J. Thielen, D. Rykx, “Evaluatie van nieuwe hashfunctie kandidaten op FPGA,” M.S. thesis, Katholieke Hogeschool Limburg, Diepenbeek, België, 2009.

[3] A. Menezes, P. van Oorschot, S. Vanstone, “Handbook of Applied Cryptography,” Overview of Cryptography, CRC Press, 1996, hoofdstuk 1 + 9.

[4] M. Vandenwauver, Katholieke Universiteit Leuven, Laboratorium ESAT-Groep COSIC, “Introduction to Cryptography,” http://www.esat.kuleuven.be/cosic/intro/.

[5] J. Daemen, V. Rijmen, “The design of Rijndael: AES --- the Advanced Encryption Standard,” Springer-Verlag, 2002, ISBN 3-540-42580-2.

[6] Wikipedia, “Birthday paradox,” geraadpleegd op 20 april 2010, http://en.wikipedia.org/wiki/Birthday_paradox.

[7] A. Regenscheid, R. Perlner, S. Chang, J. Kelsey, M. Nandi, S. Paul, “Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition,” September 2009, http://csrc.nist.gov/groups/ST/hash/sha3/Round1/documents/sha3_NISTIR7620.pdf.

[8] K. Matusiewicz, M. Naya-Plasencia, I. Nikolic, Y. Sasaki, M. Schläffer, “Rebound Attack on the Full LANE Compression Function,” ASIACRYPT'09; LNCS 5912 Tokyo, Springer, 2009 (online preprint: http://eprint.iacr.org/2009/443.pdf).

[9] National Institute of Standards and Technology, Computer security division: Computer Security Resource Center, April 2005, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[10] S. Indesteege, “The LANE hash function,” Oktober 2008, http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf.

[11] B. Baldwin, A. Byrne, M. Hamilton, N. Hanley, R. P. McEvoy, W. Pan and W. P. Marnane, “FPGA Implementations of SHA-3 Candidates: CubeHash, Grøstl, LANE, Shabal and Spectral Hash,” 2009, http://eprint.iacr.org/2009/342.pdf.

[12] Ö. Küçük, “The Hash Function Hamsi,” September 2009, http://www.cosic.esat.kuleuven.be/publications/article-1203.pdf

[13] J. Fan, “Hardware Evaluation of The Hash Function Hamsi”, http://www.cosic.esat.kuleuven.be/publications/article-1322.pdf.

[14] R. Benadjila, O. Billet, H. Gilbert, G. Macario-Rat, T. Peyrin, M. Robshaw, Y. Seurin, “SHA-3 Proposal: ECHO,” Oktober 2008, http://crypto.rd.francetelecom.com/echo/doc/echo_description_1-5.pdf.

[15] C. De Cannière, H. Sato, D. Watanabe, “Hash Function Luffa”,Oktober 2009, http://csrc.nist.gov/groups/ST/hash/sha3/Round2/documents/Luffa_Round2_Update.zip.