9th IFAC Workshop on Programmable Devices and Embedded Systems
Roznov pod Radhostem, Czech Republic, February 10-12, 2009

Hardware implementation of the MD5 algorithm.

D. Pamula*, A. Ziebinski*

*Silesian University of Technology, Gliwice, Poland (e-mail: danuta.pamula, adam.ziebinski@polsl.pl).

Abstract: The paper presents a hardware implementation of the MD5 hash generator. The generator comprises MD5 processing unit, data exchange interface and memory block for storing input messages. A general concept and implementation of the MD5 generator modules is described. Presented solution is based on the finite state machines; its performance is compared with other MD5 implementations. The MD5 generator has been modelled using VHDL and targeted to a Xilinx Spartan-3E device. Unlike compared implementations, which are mainly designated for Internet applications, this module is optimised to work as a part of video based device for measuring road traffic parameters. The MD5 generator is responsible for signing video stream captured in real time.

Keywords**:** MD5, hash functions, cryptography, FPGA.

1. INTRODUCTION

Cryptographic hash functions became very popular and nowadays are used in many different domains requiring information exchange. Mostly they are applied in securing digital information, which in Internet era is one of the most important issues. Operations such as banking transactions, verification of digital signatures, authentication and generally sending of any data via Internet must provide very high security of information exchange.

The role of hash functions is to protect the data with a digital fingerprint. To uniquely identify the input data the hash must have the following properties: it must be easy to compute, it must be very hard to produce the input message out of given digest, and it must be infeasible to find distinct input strings which hash to the same value. The most popular, widely used, hash functions are MD5 (Message-Digest 5), SHA (Secure Hash Algorithm) family of functions and RIPEMD group (B.Schneier (2002)).

Commonly, hash algorithms are used to secure data and to protect data integrity in Internet applications, which is not very speed demanding. This paper presents a hardware solution, which could be used for calculating MD5 signature of video stream captured in real time. The designed architecture of the MD5 algorithm generator module is incorporated into video-detector hardware providing message digests for a stream of images registered in real time. It is modelled using VHDL. The target device for this implementation is Spartan3-E Field Programmable Gate Array (FPGA) manufactured by Xilinx.

The FPGA chips offer great flexibility for the hardware designers. As the technology progresses the offered logic block densities of the devices increase allowing for embedding bigger and bigger processing structures. The chips manufacturers beside logic fabric provide also dedicated, circuits for multiplication, for managing clock signals (PLL and DCM) and for implementing memories (Block RAM, Memory Blocks) in the devices (Xilinx, Altera). The FPGAs provide also resources for high-speed solutions. The low-cost devices support up to 300 MHz clocking and advanced provide clocking of up to 550 MHz. Benefiting from this feature MD5 hash function module may be constructed as fast as needed being constrained only by the algorithm structure not by the hardware.

The practical implementation of the generator is very demanding and a way of complying with the requirements is to provide a compact and fast enough implementation and incorporate it in the already developed architecture.

The most secure hash functions according to the National Institute of Standards and Technology are now those of SHA (Secure Hash Algorithm) group. The MD5 algorithm is considered to be less secure but it is fast and still sufficient to be used for less demanding applications than for example banking transactions. In case of the video-detector the MD5 was considered to be the most suitable for the purpose.

1.1 Video-detector device

Modern traffic control algorithms require reliable and real-time traffic information. This information has to be collected by vehicle detectors installed at intersections. The most common detection systems use inductive loops or video cameras. Video detection becomes more and more popular because it is cost effective and offers a number of advantages such as the ability to track vehicles and provide complex data on road traffic.

The domain of image processing demands the highest processing capabilities possible but at a cost acceptable for the end user. Road traffic authorities are interested in video surveillance projects at costs comparable to traditional traffic data collection systems. This requirement is hard to comply with using ordinary video processing equipment. A way to meet this requirement is to use specialized processing hardware. Such hardware extensively utilizes parallel

processing. The problem of concurrent execution of different processing tasks can be solved using multiple processing units working parallely on a set of data or feeding a pipeline of processors with parts of data, which make up the original data set (W.Pamula (2008)).

The first solution is highly flexible, but unnecessarily complicated as image processing can be carried out efficiently, using simple processing elements working on small pixel neighbourhoods.

The second solution based on partitioning of the data for processing can be implemented very effectively using gate arrays. Gate arrays especially FPGA, which can easily be reconfigured, allow for efficient design of multiple simple processing blocks.

In the Department of IT Systems in Transport at the Faculty of Transport of the Silesian Technical University a R&D project was undertaken to develop a series of video based devices for measuring road traffic parameters. The measurement tasks were grouped and the following constructions of video detectors were proposed:

- WD-P – vehicle detector,
- WD-K – vehicle counter and classifier,
- WD-M – traffic parameters measuring device,
- WD-R – road incidents recorder.

A common processing platform was designed. The platform is based on a FPGA. Each measurement task requires a specific FPGA structure, which is created using a configuration set.

The MD5 module is used in the recorder part of the project. It is responsible for signing the video stream coming from the CCTV camera, before it is stored.

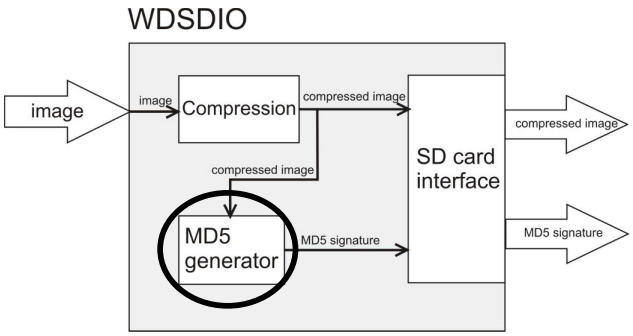

Fig 1. MD5 generator in video-detector.

The detector consists of following modules: WDKAM (video camera interface), WDRAM (interface to DDRAM memory), WDCAN (CAN interface module), WDSPI (SPI interface), WDUSB (together with WDSPI the modules provide programming utilities for the FPGA), and WDSDIO (SD card interface. More details on each of the enumerated modules can be found in Project Report (2007). The MD5 generator is a part of WDSDIO module. Inside details are presented on fig.1. The MD5 generator is marked with oval on the figure.

The whole process of recording the images works as follows: first, when red light appears on the traffic lights, the image is fetched from the CCTV camera, analysed and then processed by other modules. Next the image is compressed and signed with MD5 hash. Finally the compressed image and its hash are written to SD HC card.

2. MD5 ALGORITHM OVERVIEW

The MD5 hash function generates 128-bit message digest from an arbitrary, finite bit-length input message. The algorithm processes 512-bit blocks further divided into 32-bit words. It consists of 64 rounds. It is an extension of the MD4 algorithm, which is slightly faster and consists of only 48 rounds, but is also proven to be less secure. The MD5 algorithm works as follows.

Lets assume we have b-bit input message M. Length b can be of arbitrary but finite size. First we have to pad (extend) the message, because its bit-length needs to be congruent to 448 modulo 512. The padding is as follows: first we append single '1' bit to the end of the message and then, '0' bits to achieve the desired length. Message extension is always performed even if M is already congruent to 448 modulo 512. At least one and at most 512 bits are appended.

After the padding step the 64 bit representation of the length of the M is appended to the end of the message. We append initial b coded on 64 bits. In case when b is greater than 2^{64} only the low order bits of length representation are used.

After this initial pre-processing the length of the input data is equally divisible by 512. The message is then divided into 512-bit blocks: $M = \{m_1, m_2, m_3, \ldots, m_n\}$ and further m_i is divided into sixteen 32-bit words $m_i = \{m_{i,1}, m_{i,2}, m_{i,3}, \ldots, m_{i,16}\}$.

To calculate the hash value of M four additional 32-bit registers A, B, C and D are needed. Those registers at the beginning are loaded with predefined initial values, that is:

A = x"01234567", B = x"89ABCDEF",

C = x"FEDCBA98", D = x"76543210".

Before the start of processing of the message four auxiliary logic functions F, G, H and I are defined.

$$F(X,Y,Z) = (X \cap Y) \cup (\overline{X} \cap Z), \quad (1)$$

$$G(X,Y,Z) = (X \cap Z) \cup (Y \cap \overline{Z}), \quad (2)$$

$$H(X,Y,Z) = X \oplus Y \oplus Z, \quad (3)$$

$$I(X,Y,Z) = Y \oplus (X \cup \overline{Z}), \quad (4)$$

where X, Y, Z are 32-bit words, $\cap$ stands for logical and, $\cup$ is logical or, $\oplus$ stands for xor operation, and $\overline{X}$ evaluates to not (X). All operations are bit wise. Functions (1), (2), (3), (4) take as arguments 32-bit words and return 32-bit words. During all computations the convention low order byte first is used. The above values of A, B, C and D registers also follow this convention.

The algorithm consists of 4 stages, each of which comprise of 16 rounds. Hence, 64 rounds are performed, for each 512-bit block. The MD5 works as presented on the diagram in fig. 2.

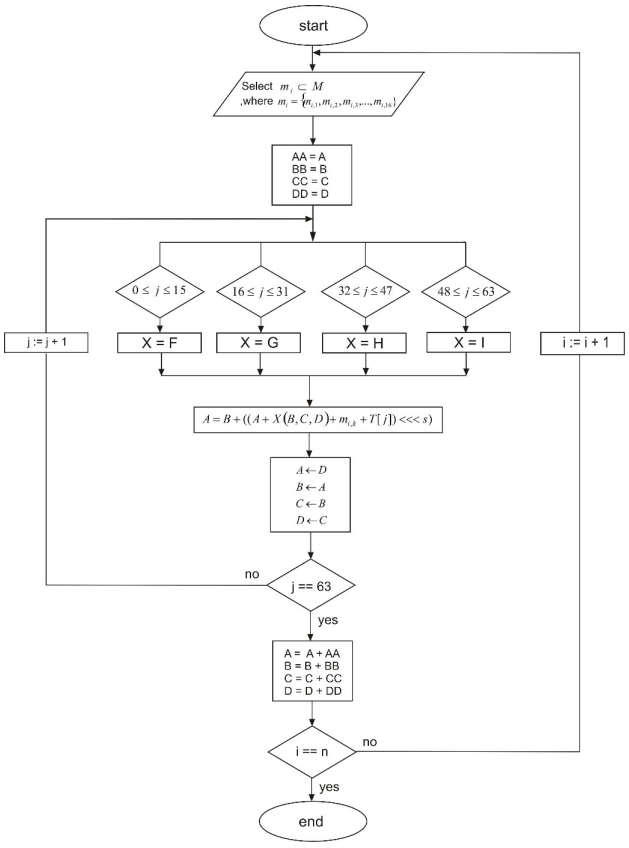

Fig 2. General architecture of MD5 algorithm.

First the 512-bit block of M is selected. Before the message block processing starts the initial values of A, B, C and D registers are stored. This is done once per 512-bit block.

Next in accordance to round number, the function *X(B,C,D)* is chosen. After function selection the operation $A = B + ((A + X(B,C,D) + m_{j,k} + T[i]) <<< s)$, (7), where A, B, C and D are above-mentioned 32-bit registers, *X(B, C, D)* represents auxiliary functions F, G, H, I, $m_{j,k}$ is k-th 32-bit word of j-th 512bit block, *T[i]* is i-th value of T, and <<< s means circular shift left by s, is performed. T is a table, which comprises of 64 elements constructed from the sine function and s is a set of 16 different shift values. Then values of the auxiliary registers A, B, C, D are exchanged and round number is incremented. This scheme is repeated 64 times. After the last round initial values stored in AA, BB, CC, DD and new values of A, B, C, D are added. The addition result is the new initial value for the registers, used in next 512-bit block processing. When all 64 rounds are performed the algorithm is repeated for the subsequent message block. After all 512-bit blocks of M are processed to obtain the message digest we have to concatenate contents of A, B, C and D registers. The low order byte of A is the least significant byte of hash value and the high-order byte of D is the most significant byte.

3. MD5 GENERATOR ARCHITECTURE

The following assumptions were made:

- Message processing should be fast enough for calculating MD5 signature of video stream captured in real time.
- Processed messages are of fixed length.
- The 512-bit blocks are initially stored in Block RAM and fetched to the module before each block is processed. Practical implementation incorporates an additional module for filling the Block RAM while the blocks are processed. In the demonstrational version the Block RAM initially contains the whole input message.

The designed module is structured as shown on fig 3.

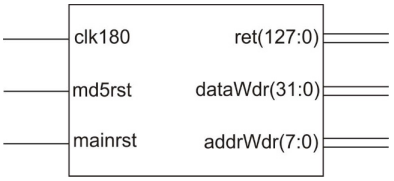

Fig 3. MD5 generator module.

It has three input signals: clock signal *clk180*, *md5rst* and *mainrst*, which together comprise the reset for the module. The clock controlling the synchronisation in the module is as noted previously 127 MHz but annotated as *clk180*.

There are two in-out ports, *addrWdr* and *dataWdr*, for exchanging data with the memory block. The signals are declared as ports to be able to connect external memory blocks. The current implementation does not use these, because memory is declared inside the module. The option of connecting external memory blocks is required by the video-detector structure. The data storage in demonstrational implementation is a single port Block RAM. In practical implementation it is substituted with dual port Block RAM. Block RAMs are structured of dedicated memory blocks available on target FPGA platform. The memory block, used for storing input messages for MD5 algorithm generator, is structured as follows. The block depth is 192 and the words stored are 32-bit long. Together the block accommodates 768Bytes, which is one line of the captured video stream.

The port on which the message-digest is outputted, is called *ret,* it is a 128-bit bus. The internal structure of MD5 generator current implementation is presented on fig.4.

3.1 Controller block

The MD5 algorithm generator solution consists of the following parts: interface to the MD5 algorithm solution, interface to Block RAM and MD5 algorithm solution. The controller block on fig. 4 represents both interfaces. Its main functions are to provide data for the algorithm, and to start and restart the algorithm processing.

The controller is designed as the finite state machine called WDR (video-detector RAM) FSM in accordance to its main function that is providing the data for the algorithm. It takes sixteen 32-bit words from the Block RAM and stores them in an array. Thus the array contains 512-bit block required by

the algorithm divided into 32-bit chunks. Block RAM is interfaced using three signal paths: 8 bit address bus (*addrWdr*), 32 bit data bus (*dataWdr*) and clock signal (*clk180*). MD5 module is restarted with *md5rst* signal set to '1'. WDR FSM is also restarted when the main reset signal is pulsed. The structure of the WDR FSM is shown on figure 5.

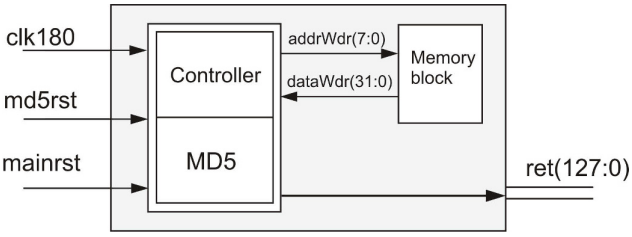

Fig 4. Internal structure of MD5 generator

In the beginning the counters used for indexing the memory and for indexing the array are loaded. Next their values are assigned to the indices (memory address line and array index). When both counters are loaded and the RAM interfacing signals are initialised the state-machine goes to *get_512bits* state, in which as the name implies the process of storing the 512–bit message block in an array is performed.

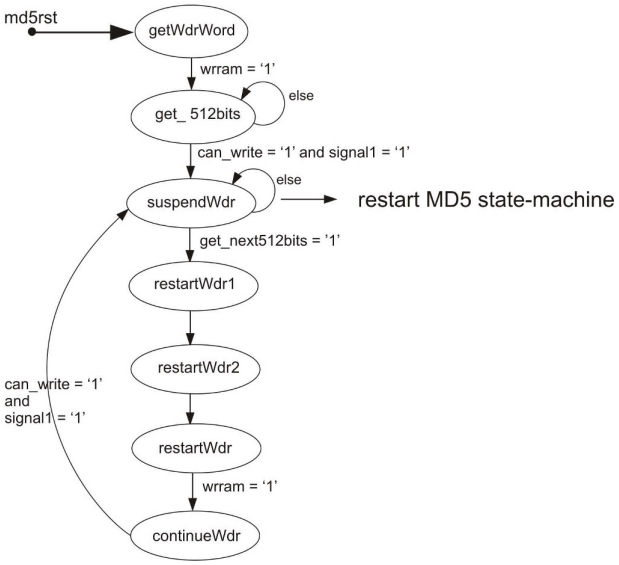

Fig 5. Structure of WDR FSM

The state also restarts the MD5 FSM, which processes stored data block. The request for data issued from MD5 FSM is denoted by setting *get_next512bits* signal to '1'. When the signal is set, the process of re-establishing the connection with memory starts. In three subsequent states: *restartWdr1*, *restartWdr2* and *restartWdr* the counters responsible for addressing memory and the array are loaded with appropriate value and the memory address and data bus are prepared for communication. Finally the state-machine reaches the *continueWdr* state, which performs operation similar to the one conducted in *get_512bits* state; that is it supervises transfer of words from memory to the array. When the transfer is finished the state-machine arrives again in *suspendWdr* state waiting for the request for data.

After a number of simulation rounds and implementation corrections a flawless output was generated. To make sure that the FSM works correctly additional test were carried out with random memory contents and these proved successful

3.2 MD5 algorithm block

The specific feature of most common hash algorithms is their simplicity. Basic recommendation imposed on one–way algorithms designers is ease of computation. Thus the architecture of the MD5 does not require any complex gate structures and special synchronisation.

Initially it was assumed that all operations of MD5 algorithm should last at most one clock cycle. Hardest to fit in this time schedule were the operations of 32-bit word additions and the circular shift operation. The concept of implementation had to be revised to account for complexity of additions and circular shift operation. For more details about the concept see D.Pamula (2008).

The nomenclature used in the implementation slightly differs from the one presented in second section to describe the algorithm. Throughout this section the notion round will be denoted as step and stage as round.

The MD5 algorithm solution is also based on the state machine. The FSM starts when *startMD5step* signal is pulsed (see fig.6). The reset of MD5 FSM is controlled by WDR FSM. There are three initialisation states: *begin_step0*, *begin_step* and *set_step_cnt*. First state is performed once per message, second once per 512-bit block of the message and third, once per 32-bit chunk of the processed 512-bit block. In the first state initial values are assigned to A, B, C and D registers, in the second the step counter is zeroed and contents of A, B, C, D are stored in temporary registers AA, BB, CC, DD. In the third state the value of the step counter is set. In the following states *choose_round*, *add_const2* value of $(A + X(B,C,D) + m_{j,k} + T[i])$ is evaluated. Addition of first two factors is controlled by the state of MD5 FSM due to the fact that function *X(B,C,D)* changes according to the step counter. The addition of last two factors is not dependent on the state machine.

In the state *choose_round* the *X(B,C,D)* function is selected. Then it is evaluated and to the result the contents of A register are added. The *add_const2* state is responsible for final addition of the two achieved results. The components of the additions are represented in the implementation by the following signals: A, which is represented by a 32-bit vector; *X(B,C,D)* represented by *functionBCD* signal; $m_{j,k}$, which is an element of *fromWdr* array; *T[i]* a constant from T array. The 32-bit word of the message, $m_{j,k}$, is selected using index k stored in an array. Index k required in the step is determined by the step counter. Elements from T array of constants are selected also according to the value of step counter. It should be noted that all additions are modulo 2^{32} additions.

When components of the equation are added the circular shift operation is performed which is controlled by the *setA* state.

Taking speed assumption into consideration, the circular shift was implemented as a barrel shifter. The barrel shifter instead of shifting the register performs the rearrangement of bit vector. Although it requires more FPGA resources it needs only one clock cycle to rearrange the register. The amount of shifts, which need to be performed, is represented by constant s. The constant is stored in an auxiliary array and selected according to the step counter.

When the evaluation of the main MD5 algorithm operation $((A + X(B,C,D) + m_{j,k} + T[i]) <<< s)$ is finished, two succeeding states: *setA*, and *setB* reassign values of A, B, C and D registers. In order to avoid declaration of the auxiliary signals the reassignments are done as follows. First contents of A are overwritten by the contents of D register, then contents of D are substituted with C register contents, next value of C is replaced by the value of B. Finally in *setB* state the result of addition and shifting are added to the old value of B and assigned to the B register.

Eventually the state-machine goes either to *last_step* or *inc_step_cnt* state depending on the value of step counter. *Inc_step_cnt* state increments value of round counter and restarts the processing of the block (the FSM goes to *set_step_cnt* state). *Last_step* state determines if the hash value calculation is already finished, or if the block created from extension bits must be processed, either if the new block of data is needed.

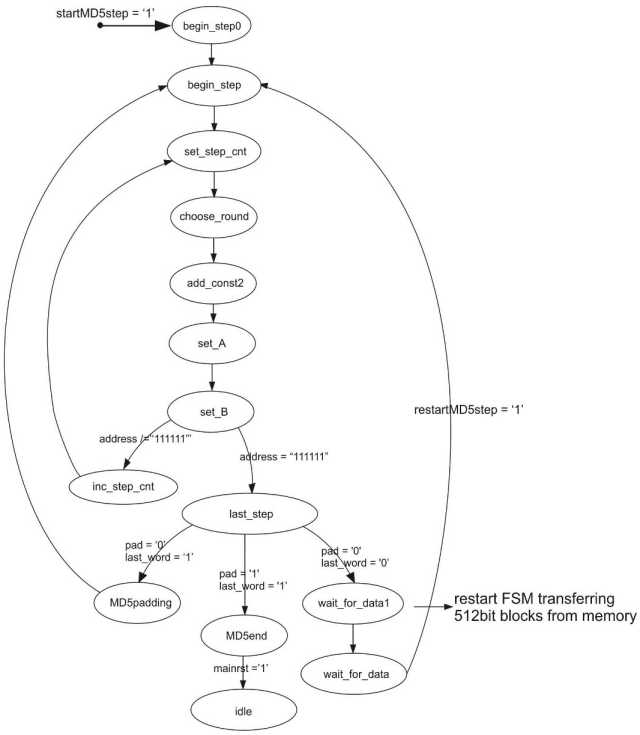

Fig 6. MD5 FSM

First option ends calculation of the message digest, the FSM reaches *MD5end* state, in which the hash value stored in A, B, C, D registers is outputted to the *ret* vector. The least significant byte of A is the most significant byte of *ret* and the most significant byte of D is assigned to the least significant byte of *ret*.

When second option is evaluated to true, the FSM goes to *MD5padding* state, which starts the processing of last block. The last block, which is created from padding bits, is stored in the P array not in the memory block. It can be done because the length of input message is constant. The state succeeding *MD5padding* is *begin_step*.

Third option selects *wait_for_data1* state, which restarts the WDR FSM in order to provide next message block for MD5 FSM. Then the FSM goes to *wait_for_data* state in which it stays until the next message block is stored in the array and *resetMD5step* signal is set. Eventually it goes to *begin_step* and hash calculation is continued until the last block is processed. The MD5 FSM structure is presented in fig.6.

After the implementation and simulation the performance tests were carried out to evaluate if the solution is suitable for the video-detector platform. To stabilise the solution and to be less dependent on the Place and Route tool the placement of the module was constrained. The whole MD5 generator was confined to a box of CLB's in one of the clock quadrants of the FPGA.

4. RESULTS OF THE IMPLEMENTATION

The architecture described is coded in VHDL using Xilinx ISE 9.2i environment. It was tested with messages of different sizes and contents. All the simulations were performed in ModelSim tool and on the video-detector board.

First discernible difficulty during the implementation was the addition of 32-bit words. In the main operation of the algorithm an addition of even as many as four 32-bit words at a time is required. This type of addition depends on the performance of carry logic modules (CLM) and may lead to unpredictable results when carry chain is too short. To avoid unpredictable results the additions are divided into additions of two words at a time, hence instead of having four 32-bit input adder there are two faster two input adders. Another problem was where to store constant values to have them fetched instantly; storing constants in arrays - addressable memory structures, solved that. Next difficulty was to derive a circuit for exchanging contents of the A, B, C, D registers in order not to overwrite the needed value. However during implementation it occurred that the exchange of the register values is the least problematic task. The circular shift was also a very demanding problem, because it should be fast enough to satisfy overall timing constraints. Thus it could not be implemented as simple shift register instead barrel shifter was used, as described above.

The performance tests had shown that increasing the number of states of the MD5 FSM increases the possible operating frequency, however then the throughput may fall below the desirable value. On the other hand decreasing the number of states decreases the frequency but not always increase the throughput. The final solution of the MD5 FSM was very scrupulously tested optimising the number of states, which could be merged to give the best performance.

Implementation	Device	Frequency	Throughput	Size
This design	Spartan3-E XC3S1200E-4	125MHz – 130MHz	166 Mbps – 173 Mbps	852 slices
	Virtex V XC5VLX30-3	365 MHz	486 Mbps	301 slices
J.Deepakumara, et al.	Virtex VI000FG680-6	21 MHz - 1.4 MHz.	165 Mbps - 354 Mbps	880 slices – 4763 slices
Chiu-Wah Ng, et al.	Altera EPF10k50sbc356-1	26.66 MHz	206 Mbps	1964 LC
Kimmo Järvinen, et al.	Xilinx Virtex-II XC2V4000-6	75.5MHz - 93.4MHz	586 Mbps - 725 Mbps	647 slices- 7997 slices
T.S. Ganesha, et al.	Xilinx Virtex2P	58.137 MHz - 104.875 MHz	202 Mbps - 331 Mbps	2837 slices – 2971slices

Table 1. Comparison of MD5 algorithm implementations.

5. PERFORMANCE COMPARISON

The main advantage of the designed solution is that it can operate at high frequencies. When implemented in Spartan3-E FPGA it works with frequencies up to 130MHz. However in Virtex2 chip, in which most of the compared architectures are implemented it can reach 174 MHz and for Virtex5 even 365 MHz.

The assumed throughput of the whole generator according to the video detector specification should be at least 13.5 MBps (108 Mbps). After simulation the throughput of the whole module, including data fetching, at frequency 125MHz is 17MBps (136Mbps). The throughput of the MD5 algorithm solution is estimated using the formula:

throughput = (block size *x* clock frequency)/no.of cycles (8)

The module needs 384 cycles to process one 512-bit message block. The assumed working frequency is 125MHz thus the throughput is 166Mbps. The area utilization is 852 slices (87,111 gates), which is 9% of available slices in the target FPGA. Table 1 presents comparison of other known MD5 algorithm implementations with the one described here.

6. CONCLUSIONS

The presented solution meets the assumptions set and complies with the requirements of the video-detector device. It is capable of hashing incoming video stream in real time. The speed of the incoming video frames implies the throughput of the MD5 generator. If the total throughput is greater than the speed of the video stream it is possible to hash it in real time. As stated in previous section the achieved throughput of the whole module (including data fetching) is 17MBps, which is enough to satisfy the performance speed requirement implied.

As mentioned in the previous sections the video detector device incorporates more than a single MD5 generator solution hence the module should be also compact. It cannot incorporate too much resources of the FPGA. The more the chip area is occupied the harder to configure it properly. After comparison with other implementation it occurred that it is one of the most area-optimised solution. It takes only 852 slices of the FPGA. Further, it may be attempted to increase the solution throughput by for example fetching next message blocks while the previous blocks are being processed. However this requires more analysis of the video camera interface and captured video stream.

Summarising, although the solution has not the highest throughput it can operate in the highest frequencies and accommodates the smallest number of slices. It is very compact solution and suits best for its application.

REFERENCES

J.Deepakumara, H.M.Heys, R.Venkatesan. (2001). FPGA implementation of MD5 Hash algorithm, In Canadian Conference on Electrical and Computer Engineering, vol 2, pages 919-924.

T.S.Ganesh, M.T.Frederick, T.S.B.Sudarshan, A.K.Somani. (2007). Hashchip: A shared-resource multi-hash function processor architecture on FPGA, In INTEGRATION, the VLSI Journal, vol.40, pages 11-19.

K.Järvinen, M.Tommiska, J.Skyttä. (2005). Hardware Implementation Analysis of the MD5 Hash Algorithm, Proceedings of the 38th Hawaii International Conference on Science.

C.-W.Ng, T.-S.Ng, K.-W.Yip. (2004). A unified architecture of MD5 and RIPEMD-160 Hash algorithms, In IEEE International symposium on Circuits and Systems

D.Pamula. (2008). Implementation of MD5 hash algorithm in FPGA. Master thesis dissertation, Silesian University of Technology, Gliwice 2008

W.Pamula. (2008). Issues of hardware implementation of image based vehicle detection, In Central European Conference on Information and Intelligent Systems 2008

Project Report (2007): Modules of Video Traffic Incidents Detectors ZIR-WD for Road Traffic Control and Surveillance. WKP-1/1.4.1/1/2005/14/14/231/2005, Katowice, Poland 2007

B.Schneier. (2002). Applied Cryptography. Wydawnictwa Naukowo- Techniczne, Warszawa 2002.

Xilinx, Altera. URL, www.xilinx.com, www.altera.com.