FPGA HW Accelerator of the First Step of Systematic Two-Level Minimization of Single-Output Boolean Function

Branislav Madoš, Norbert Ádám, Zuzana Bilanová, Martin Chovanec

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
e-mail: {branislav.mados, norbert.adam, zuzana.bilanova, martin.chovanec}@tuke.sk

Abstract: Boolean function minimization is an area important not only in the development and optimization of digital logic, but also in other research and development areas, such as, the optimization of control systems, simplifying program logic, artificial intelligence, etc. The aim of this paper is to present a hardware accelerated first step of the systematic minimization of single-output Boolean functions – the generation of a set of prime implicants for both the disjunctive normal form (DNF) and the conjunctive normal form (CNF), having defined the OFF and ON sets and – alternatively – also the DC ("don't care") set. The proposed hardware accelerator is designed as combinational logic, described in VHDL. Its advantages include an extremely short prime-implicant-generation time in the order of ns and/or tens of ns – in case of Boolean functions with small amount of input variables – and the possibility to generate the valid-prime-implicant set of Boolean functions having a defined number of input variables at a constant time, regardless of the cardinality of the ON or, eventually, the DC sets. However, these advantages come with a large spatial complexity – the number of utilized implementation elements – of the respective combinational module, generating the prime-implicant set. The authors verified the proposed design using Field Programmable Gate Array (FPGA) technology, implementing the hardware using a Xilinx Kintex-7 KC-705 Evaluation Kit development board.

Keywords: Boolean function minimization; prime implicant generation; combinational logic; FPGA; disjunctive normal form; DNF; conjunctive normal form; CNF; hardware accelerator; systematic minimization; heuristic minimization

1 Introduction

Boolean function minimization is a significant problem not only in academia and scientific research, but also in many research and development areas. This includes, for example, the development of logic designs, such as Programmable Logic Array (PLA) technology, Field Programmable Gate Array (FPGA) technology, Application Specific Integrated Circuit (ASIC) technology, as well as the design and development of control systems including the vast and important domain of controlling intelligent buildings and houses [1], software engineering (to optimize logic used in software), artificial intelligence, security of computer systems [2] and many others. Boolean function minimization is a significant challenge mainly if the input variables are numerous (counting hundreds or thousands), rendering many minimization approaches impractical, since these cannot provide minimization using available hardware in considerable, practical time.

Today, logic design optimization may be classified by various criteria, such as design characteristics (i.e. combinational logic or sequential logic), the amount of levels (two-level or multi-level minimization), or the implementation method (algebraic, table-based or graphic minimization). Some algorithms are based on using human expertise in finding patterns, thus these are implemented “manually”. Further approaches are algorithmic – these implement the respective algorithms in software running on the CPUs and GPGPUs of traditional computers [3] [4] [5].

Another way of classifying optimization is to take the minimality of the solution into account – in this case, the categories are systematic and heuristic minimization. Systematic minimization will always find the minimum solution for the specified minimization criteria. The most famed approaches of systematic minimization include graphical minimization using Karnaugh maps (KM) and a tabular method using the Quine-McCluskey (Q-M) algorithm. On the contrary, heuristic minimization often yields a near-minimal solution, with an advantage: a cut (often radical) in processing time and resources. Thus, the aim of using heuristic minimization is to utilize it even in case of Boolean functions having high amounts of input variables, in case of which systematic minimization would not be of any practical use. The most famed solutions of heuristic Boolean function minimization include Espresso – the de-facto industry standard in Boolean function minimization – and its derivatives, as well as the BOOM and BOOM-II algorithms, respectively. For a discussion of both systematic and heuristic minimizations see section 2 herein.

In this paper, the authors focused on the field of systematic, two-level minimization of single-output Boolean functions – when implementing the algorithm, instead of using the CPU and/or the GPGPU to write software, they chose to implement the algorithm in a hardware-accelerated form, using Field Programmable Gate Array (FPGA) technology. The proposed solution is based on previous development – in [6], the authors presented a hardware-accelerated generator of prime implicants for single-output Boolean functions, based on combinational logic. In this paper, the authors describe an enhancement of the aforementioned solution, allowing the generation of prime implicants both the disjunctive (DNF) and conjunctive (CNF) normal form; compared to the previous version, the solution proposed herein allows the definition of not only of the OFF and ON sets, but also of the DC (don't care) set. The aim was to create a circuit that would significantly minimize the prime-implicant-generation time to the order of ns and/or tens of ns in case of Boolean functions with small amount of input variables.

The contribution hereof lies in the following:

- Design of a hardware-accelerated implementation of the first step of the systematic two-level minimization of single-output Boolean functions, based on a combinational logic module, to generate prime implicants; the proposed hardware accelerator allows processing of Boolean functions with output values defined not only by means of OFF and ON sets, but also the DC set.

The structure of the paper is as follows:

Section 2 deals with the related work in the field of systematic and heuristic Boolean function minimization. Due to the abundance of papers published in this field, the authors resorted to a selection of the fundamental works.

Section 3 contains a detailed description of the proposed hardware accelerated Boolean function minimizer. In the introductory part of this section, the authors describe the encoding of the hardware accelerator's input and output vectors. In the last part of the section, the authors describe the structure of the hardware accelerator itself, split into three submodules: the prime-implicant-generation mode selection module; the prime-implicant-generation module (implemented as a combinational logic circuit); and the invalid-prime-implicant-exclusion module.

Section 4 summarizes the testing results of the hardware-accelerator (implemented using a Xilinx Kintex-7 KC-705 Evaluation Kit evaluation board) for various numbers (2 to 8) of input variables of single-output Boolean functions.

Section 5 contains the conclusions, distilled from the results of the implemented tests, described in the previous section.

2 Related Work

Due to the large amount of work published in the field of systematic and heuristic Boolean functions minimization, this section of the paper contains only the selection of papers that are representing fundamental works related to the solution designed as the part of this work and presented in this paper.

Systematic minimization. In 1881, Allan Marquand presented his diagrams, which allowed the simplification of the graphical presentation of Venn diagrams for a larger number of variables [7]. In 1951, the Harvard minimizing charts were presented by Howard H. Aitken, described in detail in [8]. In 1952, Edward Westbrook Veitch developed a Boolean function minimization method [9], along with the corresponding diagrams, often called Marquand-Veitch diagrams. This method was later perfected by Maurice Karnaugh in 1953 [10] – today, it is known as Karnaugh maps (KM or K-maps). In 1956, Svoboda created graphical aids for systematic Boolean function minimization [11].

Karnaugh maps, sometimes referred to as Karnaugh-Veitch (KV) maps. These are not only a graphical notation for Boolean functions, but mainly serve for minimization purposes. These utilize human expertise in finding patterns within the graphical representation of the Boolean function depicted as a diagram, instead of minimizing the particular Boolean function using a computer program. to represent Boolean functions, Karnaugh-maps use a two-dimensional grid containing 2^n fields, n being the number of input variables. The fields are organised as a $2^k \times 2^l$ grid, where $k + l = n$ and k differs from l by at most 1. Each field of the Karnaugh-map contains information about the particular Boolean function's output value. A limitation of this method is that visual pattern matching and the subsequent simplification in K-maps is practical only for a very small number of input variables, while the limit amount is stated to be 5–6 input variables. A further drawback is the human factor, which may introduce errors into the process.

The Quine-McCluskey method, also referred to as the Q–M method, is a tabular method of systematic Boolean function minimization, which is, in terms of the achieved results, analogous to the K-map method. It was developed in 1952 by Willard Quine and Edward McCluskey [12] [13] as a two-step method. In the first step, the algorithm generates the prime implicants of the Boolean function, while in the second step, it solves the issue of covering the Boolean function by the prime implicants. Compared to the K-map method, the advantage of this method is that it does not rely on the capacity of a human to find patterns, but rather it introduces an algorithm ready to be implemented in a computer, thus it may be used to process Boolean functions with significantly more variables. The systematic approach of this method prevents its practical use in case of high amounts (i.e. hundreds or thousands) of input variables – this method is time and resource hungry.

Heuristic minimization. The MINI heuristic minimizer was presented by Hong et al. in 1974 [14]. It generates a solution without the necessity to generate all prime implicants of the Boolean function to be minimized. The Espresso logic minimizer was presented by R. K. Brayton et al. with the goal to minimize logic circuits using heuristic methods [15]. The Espresso-MV (Multi-valued) method is a derivative of the Espresso method; it was developed in 1986 by Richard L. Rudell. Both the heuristic and systematic minimization approaches were described in [16].

The C language source code of the Espresso algorithm is available in [17]. Further improvements to the Espresso method include the Espresso-Exact and Espresso Signature methods [18].

The two-level Boolean minimization tool called BOOlean Minimizer (BOOM), developed by Hlavička and Fišer, is based on the new paradigm of implicant generation: unlike other minimization methods, generating implicants using the bottom-up approach, the BOOM method uses a top-down approach. A further advantage is also in the reduction of the amount of prime implicants. The proposed algorithm is well suited for Boolean functions with the large number of variables (up to thousands), when other algorithms are not able to yield results in reasonable time [19] [20] [21] [22] [23]. The FC-Min Boolean minimizer was introduced by Fišer and Kubátová in [24]; later, it was combined with the BOOM algorithm as the BOOM-II Boolean minimizer [25] [26].

3 Proposed HW Accelerator

The hardware accelerator proposed herein uses a combinational logic circuit as its most important part, aimed at the generation of prime implicants of the Boolean function. The circuit design is described using the VHDL language. In the phase of testing the design, its practical implementation was performed using Field Programmable Gate Array (FPGA) technology.

The aim of this section is to describe the encoding of two binary input vectors – containing the OFF set and the ON set – and/or the DC set of the single-output Boolean function. Then, the description of the encoding of the output vector – representing the prime implicant set of the particular Boolean function – follows. The last part of this section contains the description of the three modules of the hardware accelerator itself: the prime-implicant-generation mode selection module; the prime-implicant-generation module (implemented as a combinational logic circuit); and the invalid-prime-implicant-exclusion module.

3.1 Boolean Function Truth Table Encoding

The input of the hardware accelerator is the representation of the truth table of the single-output Boolean function of n input variables, as 2^n-sized binary vectors.

The size of vectors results from the line-count of the Boolean function truth table. Each such line of the truth table has a binary code assigned pursuant to the input variable configuration. If the input variable is in complementary form, 0 is used, while for variables in true form, 1 is used. This binary code may be transformed to a decadic equivalent (DE), as shown in Table 1. On its input, the hardware accelerator accepts a Boolean function output value from the set $\{0, 1, \times\}$, where $\times$ is the „don't care" value, i.e. the output value has no importance.

Table 1
Generation of decadic equivalents (DE) assignment of the respective minterms and maxterms of two input variable Boolean function

DE	Binary code	Minterm	Maxterm
0	00	$\overline{x_0}\overline{x_1}$	$x_0 + x_1$
1	01	$\overline{x_0}x_1$	$x_0 + \overline{x_1}$
2	10	$x_0\overline{x_1}$	$\overline{x_0} + x_1$
3	11	$x_0 x_1$	$\overline{x_0} + \overline{x_1}$

3.1.1 Input Vector Encoding

The hardware accelerator input is encoded using two binary vectors, A and X, where A consists of 2^n bits, $A(2^n - 1{:}\,0)$

$$A = (a_{2^n-1}, a_{2^n-2}, a_{2^n-3}, \dots, a_2, a_1, a_0) \qquad (1)$$

To $\forall a_p \in A : p \in <0; 2^n - 1>$ it applies that $a_p \in \{0,1\}$

The order of bits in the A input binary vector is selected so that the bit in position p represents the output value of the Boolean function with a decadic equivalent equal to p. If the particular Boolean function output value is set to 1, the corresponding bit of the A input binary vector is set to the same value – 1. If the particular Boolean function output value having a decadic equivalent p is set to 0 or $\times$, the corresponding bit with the p position in the A input binary vector is set to 0.

The X vector also consists of 2^n bits: $X(2^n - 1{:}\,0)$

$$X = (x_{2^n-1}, x_{2^n-2}, x_{2^n-3}, \dots, x_2, x_1, x_0) \qquad (2)$$

To $\forall x_p \in X : p \in <0; 2^n - 1>$ it applies that $x_p \in \{0,1\}$

The order of bits in the X input binary vector is selected so that the bit in position p represents the output value of the Boolean function with a decadic equivalent equal to p. If the particular Boolean function output value is set to $\times$, the corresponding bit of the X input binary vector is set to the value 1. If the particular Boolean function output value having a decadic equivalent of p is set to 0 or 1, the corresponding bit with the p position in the X input binary vector is set to 0.

If the truth table of the Boolean function defines outputs only from the {0, 1} set, only the A binary input vector creates input to the hardware accelerator input and the X binary input vector bits have to be set to 0.

3.2 Prime-Implicant-Set Encoding

The output of the hardware accelerator allows us to generate the set of prime implicants of the particular n input variable single-output Boolean function. The truth table of this Boolean function, encoded in vectors A and X, acts at the input of the hardware accelerator. The output of the circuit shows the prime implicants set for DNF or CNF form, depending on the values of the corresponding control signals.

The hardware accelerator output is encoded as the O output binary vector, consisting of $3^n + 1$ bits: $O(3^n{:}\,0)$

$$O = (o_{3^n}, o_{3^n-1}, o_{3^n-2}, \dots, o_2, o_1, o_0) \tag{3}$$

To $\forall o_p \in O : p \in <0; 3^n>$ it applies that $o_p \in \{0,1\}$

The corresponding bit of the O output binary vector at its position p in the aforementioned vector shows whether the prime implicant having the p value of its decadic equivalent is or is not a prime implicant of the particular Boolean function. If the corresponding bit of the vector is set to 1, it is a prime implicant of the particular Boolean function. It is not a prime implicant of the particular Boolean function, this bit is set to 0. The decadic equivalent of 0 and 3^n is dedicated for the single-output Boolean functions producing a constant output 0 and 1 respectively, as shown in Table 3.

For the remaining decadic equivalents, one may find out the corresponding prime implicants by converting the specific decadic equivalent to a ternary code of n ternary digits (n is the number of input variables of the particular Boolean function). Then, each such ternary digit is encoded to the corresponding variable pursuant to Table 2, i.e. in DNF form, the variable in the prime implicant description is not used (if the ternary digit is set to 0), the variable is in complementary form (the digit is a 1), or the variable is in true form (the digit is a 2).

Table 2
Encoding variables in disjunctive normal form (DNF) and conjunctive normal form (CNF), depending on the digit of ternary equivalent (TE)

Ternary digit	**DNF**	**CNF**
0	□	□
1	$\overline{x_i}$	x_i
2	x_i	$\overline{x_i}$

In CNF form, the variable in the prime implicant description is not used (if the ternary digit is set to 0), the variable is in true form (if the digit is a 1), or the variable is in complementary form (if the digit is a 2). Assigning the variables to the respective digits of the ternary equivalent of the particular prime implicant to encode its description respects the order of the input variables in the truth table of the Boolean function.

A list of all decadic equivalents, their corresponding ternary equivalents and prime implicants for the DNF and CNF forms for a two-input Boolean function is specified in Table 3.

Table 3
Equivalence of decadic equivalents (DE), ternary equivalents (TE) and prime implicants (PI) for both the DNF and CNF forms

DE	**TE**	**DNF PI description**	**DNF PI**	**CNF PI description**	**CNF PI**
0	00	$\_\_\_\_$	0	$\_\_ + \_\_$	1
1	01	$\_\_\overline{x_1}$	$\overline{x_1}$	$\_\_ + x_1$	x_1
2	02	$\_\_x_1$	x_1	$\_\_ + \overline{x_1}$	$\overline{x_1}$
3	10	$\overline{x_0}\_\_$	$\overline{x_0}$	$x_0 + \_\_$	x_0
4	11	$\overline{x_0}\overline{x_1}$	$\overline{x_0}\overline{x_1}$	$x_0 + x_1$	$x_0 + x_1$
5	12	$\overline{x_0}x_1$	$\overline{x_0}x_1$	$x_0 + \overline{x_1}$	$x_0 + \overline{x_1}$
6	20	$x_0\_\_$	x_0	$\overline{x_0} + \_\_$	$\overline{x_0}$
7	21	$x_0\overline{x_1}$	$x_0\overline{x_1}$	$\overline{x_0} + x_1$	$\overline{x_0} + x_1$
8	22	x_0x_1	x_0x_1	$\overline{x_0} + \overline{x_1}$	$\overline{x_0} + \overline{x_1}$
9	100		1		0

3.3 Proposed Hardware Accelerator Module Design

The prime-implicant-generation module of the hardware accelerator for n variable single-output Boolean functions consists of 2^n prime-implicant-generation mode selection modules, the prime-implicant-generation module itself and $3^n + 1$ modules to exclude invalid prime implicants.

3.3.1 Prime-Implicant-Generation Mode Selection Module

The prime-implicant-generation mode selection module allows the user to select, whether to generate prime implicants consisting of the Boolean function outputs, where the function output is a member of the $\{0,\times\}$ set (for CNF) or the $\{1,\times\}$ set (for DNF) or the $\{\times\}$ set (to identify invalid prime implicants consisting exclusively of DC output values). Generation mode selection is possible using the $m0$ and f control signals, their effects are stated in Table 4.

Table 4
Accepted output values of the Boolean function for different $m0$ and f control signal settings when generating prime implicants for CNF, DNF forms and for identifying invalid prime implicants (IPI)

$\boldsymbol{m0}$	$\boldsymbol{f}$	**Mode**	**Accepted output values of the Boolean function**
0	0	IPI	$\{\times\}$
0	1	IPI	$\{\times\}$
1	0	CNF	$\{0,\times\}$
1	1	DNF	$\{1,\times\}$

If the f control signal is set to 1, DNF prime implicants are generated; a 0 setting of this signal indicates generation of CNF prime applicants. If the $m0$ control signal is set to 0, the accelerator shall generate information only concerning prime implicants for which the Boolean function output has always an $\times$ value (flagged as invalid prime implicants). If the $m0$ control signal is set to 1, the accelerator shall generate information only concerning prime implicants for which the Boolean function output is a 1 or an $\times$ (for DNF); or a 0 or an $\times$ (for CNF). The output of the OP module is set to 1 if the particular output value of Boolean function belongs to the accepted set of output values, as specified in Table 4.

For the hardware accelerator of an n variable single-output Boolean function the authors used 2^n of these modules to select the mode of prime implicant generation. Every pair of $a_p \in A$ and $x_p \in X$ bits of the accelerator input binary vectors (where $p \in < 0;\ 2^n - 1 >$ represents their position in the vector) is the input of the corresponding prime-implicant-generation selector module; in Fig. 1, particular inputs are denoted as the IA and IX. Further inputs of the module include the $m0$ and f control signals. The module output, denoted as OP, is a bit of the $P(2^n - 1{:}\,0)$ vector having the p position in the vector; this shows whether the particular Boolean function output shall be included in the prime-implicant-generation in the particular generation mode (the OP value is set to 1) or it will be excluded from the generation (if the OP value is set to 0).

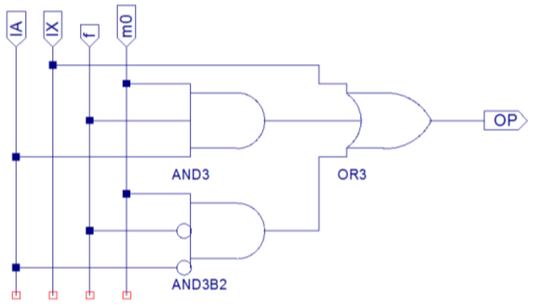

Figure 1

Combinational module for the selection of prime implicant generation mode

3.3.2 The Main Prime-Implicant-Generator Module

The input of the prime-implicant-generator module, a combinational logic circuit, is the vector $P(2^n - 1{:}\,0)$ – for a description of its computation, please refer to the previous subsection. The output of the module is the $R(3^n{:}\,0)$ binary output vector. The module consists of $l = n + 1$ layers of NAND gates, representing the potential prime implicants of the n variable Boolean function. The l_0 gate layer, containing two NAND gates, determines whether the Boolean function has a constant output value of 0 or 1. Layers l_1 – l_n contain gates representing the respective potential prime implicants (PPI). Particular NAND gate residing in layer $l_y : y \in < 1; n >$ represents the prime implicant described using y variables.

The total count of these gates in layer l_y equals to the number of potential prime implicants of the particular n variable Boolean function that may be described using y variables.

The gate in layer l_y receives information from the P input vector and from the $l_z : z \in < 1; y-1 >$ gates layers, containing gates for the potential prime implicants described by a number of input variables lower than the particular prime implicant, specifically from that part of the gates that cover the particular prime implicant. The output of the gate is set to 1 if the particular potential prime implicant is not a prime implicant of the particular Boolean function, or, to 0, if the potential prime implicant is the prime implicant of the particular Boolean function. Before constructing the R output vector, the output signal of each NAND gate is inverted to ensure that the R output vector of the module contains a bit set to 1 if the particular potential prime implicant is really a prime implicant of the particular Boolean function.

To allow a potential prime implicant to be a real prime implicant of the particular Boolean function, three conditions must be met:

- **Condition 1:** Each bit of vector P representing output values of Boolean function which are relevant for particular prime implicant, must be set to 1.

  Meeting this condition may be tested using the information gained from the P input vector of the module.

- **Condition 2: T**he Boolean function must not produce a constant value at its output.

  Meeting this condition may be tested using the information generated in the l_0 gate layer.

- **Condition 3:** The potential prime implicant must not be covered by any other prime implicant (described with a lower amount of variables).

  Meeting this condition may be verified in case of a gate in layer l_y by acquiring the information from the respective gates of layers $l_z : z < y$.

The schematic representation of the prime-implicant generator module of two-variable Boolean function is stated in Fig. 2, showing the input of the module as an input layer and three levels of NAND gates in levels 0 to 2. Layer 0 contains two gates that indicate whether the Boolean function has a constant output of 0 or 1. Layer 1 contains four gates for the potential prime implicants, interpreted for the purposes of DNF as $\bar{a}$, a, $\bar{b}$, b. Layer 2 contains four gates for the potential prime implicants, interpreted for the purposes of DNF as $\bar{a}\bar{b}$, $\bar{a}b$, $a\bar{b}$, ab.

The meaning of the respective bits of the module's R output vector is analogous to the meaning of the respective bits of the accelerator's O output vector, as stated in section 3.2 above.

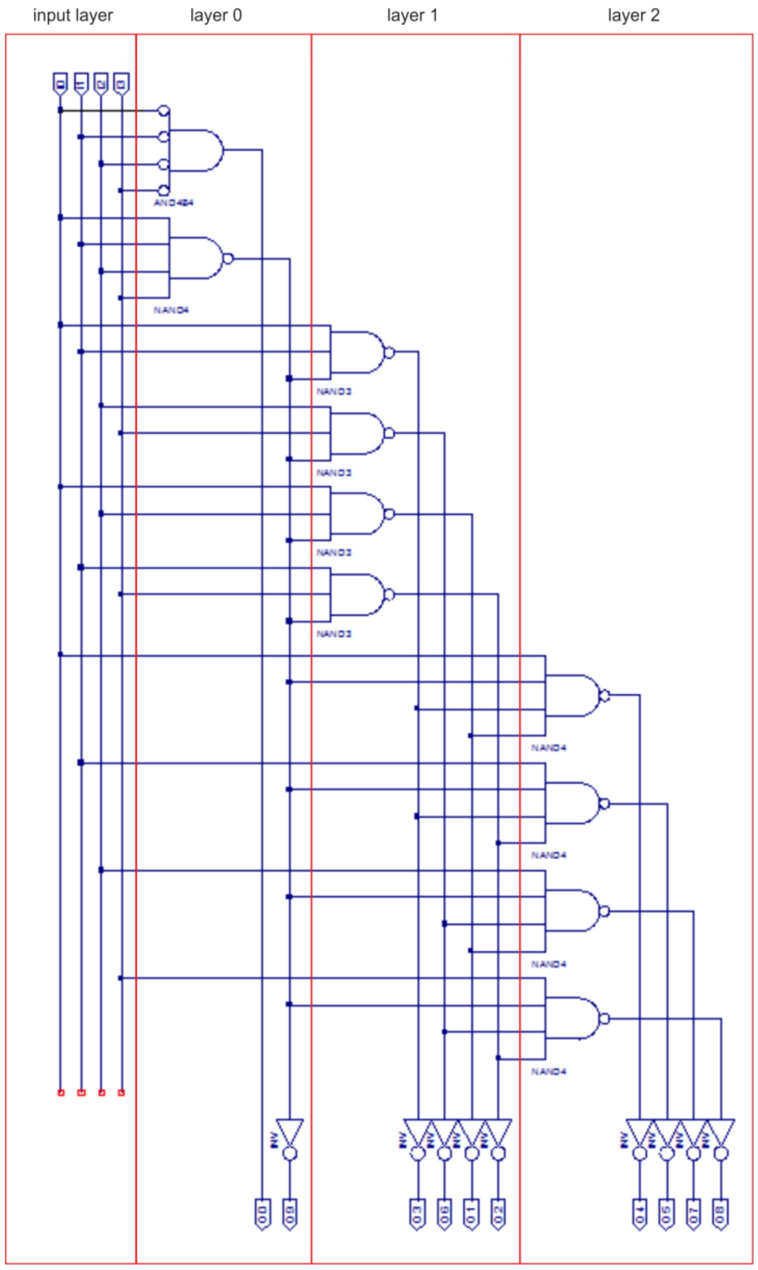

Figure 2
Gate-level schematic representation of the design of FPGA hardware accelerator module that determines prime implicants on the output of the module in the form of the *R* output binary vector for a two-variable Boolean function, represented on input of the module in the form of binary vector *P*.
Source: Madoš et al. [6]

3.3.3 Invalid-Prime-Implicant-Exclusion Module

The invalid-prime-implicant-exclusion module ensures that no prime implicant, for which the Boolean function output values belong exclusively to the DC set, is included in the final set of prime implicants of the particular Boolean function. For an n variable Boolean function, 3^n+1 modules were used to exclude invalid prime implicants. For each bit of the output vector R of the prime-implicant-generator module, such invalid-prime-implicant-exclusion module was used. The corresponding bit of the vector R at position p, is assigned to the input of the specific invalid-prime-implicant-exclusion module at the position p, brought to the input denoted as IR.

Depending on the setting of the $m0$ and $m1$ control signals, respectively, the bit of the R output vector, assigned to the IR input, is stored in the flip-flop FDE_0 (if signal $m0$ is set to 1) or in the flip-flop FDE_1 (if signal $m1$ is set to 1), as stated in the Fig. 3.

By setting signal $m0$ to 1, the circuit will generate information concerning all prime implicants, i.e. both valid and invalid. The corresponding bit at the IR input will be stored in the flip-flop FDE_0 in this case.

By setting signal $m1$ to 1, the circuit will generate information concerning invalid prime implicants, i.e. those covering the outputs of the Boolean function, in which the output is solely from the DC set. The corresponding bit at the IR input is in this case stored in the flip-flop FDE_1 and the 1 value of this bit indicates the invalidity of the prime implicant.

The module output, having the form of the O_x signal is then set to 1 only if the value if flip-flop FDE_0 is set to 1, which indicates that the potential prime implicant belongs to the set of prime implicants of the Boolean function and the FDE_0 flip-flop does not indicate the invalidity of the prime implicant. The O_x output of the module at position p is then forming the corresponding bit of the accelerator's O output vector, while the position of the bit in this vector is also p.

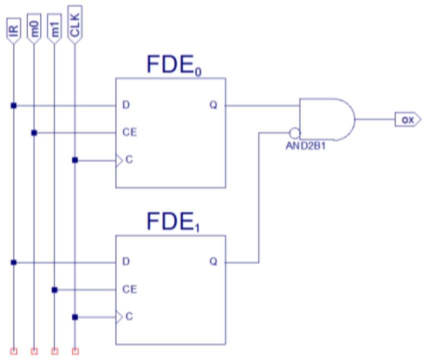

Figure 3
Schematic representation of the invalid-prime-implicant-exclusion module

If the Boolean function is defined to have an output values belonging solely to set $\{0,1\}$, the set of prime implicants of the particular Boolean function may be acquired in a single step:

Step 1: setting the $m0$ control signal to 1, the $m1$ control signal to 0 and the f control signal to 0 for CNF and to 1 for DNF, respectively.

If the Boolean function is defined to have an output values belonging to set $\{0,1,\times\}$, the set of prime implicants of the particular Boolean function may be acquired in two steps:

Step 1: setting the $m0$ control signal to 1, the $m1$ control signal to 0 and the f control signal to 0 for CNF and to 1 for DNF, respectively.

Step 2: setting the $m1$ control signal to 1 and the $m0$ control signal to 0.

With the first step, the accelerator finds out the set of prime implicants of a particular Boolean function containing valid and also invalid prime implicants (consisting solely of DC points). Therefore, the second step is executed, which yields a set of invalid prime implicants of the particular Boolean function, consisting solely of DC points. After the execution of the second step, the invalid-prime-implicant-exclusion modules ensure assembly of the output vector, containing only valid prime implicants of the particular Boolean function.

4 Results

The implementation language of the proposed modules is VHDL. As the target platform, the authors chose the use a Xilinx KC705 development board, using a Kintex-7 XC7K325T-2FFG900C series FPGA chip. The KC705 board used for synthesis has the speed grade -2 and a 2.5V LVDS differential 200 MHz oscillator. The output frequency could be changed within the range of 10 MHz to 810 MHz. The defined maximum clock speed limits the minimum response time, i.e. the time defined by the shortest clock period, in which any module implemented on the chip will work correctly (without violating the time constraints). With this FPGA chip, this value amounted to 1.23 ns.

The circuit synthesis was performed for Boolean functions with 2-8 input variables. A set of 7 top modules was created – these were implemented in VHDL using the Xilinx Vivado Design Suite HLx Edition 2016.2 development tool. The aim of testing the proposed module was to find out the hardware resource requirements of the synthesis and to measure the time required to generate the prime implicants.

Then, the authors compared the hardware resource requirements of the respective implementations of the particular top modules. The aim of the authors was to check if their expectations related to the resource consumption growth rate and response time growth rate were realistic. They expected that the resource consumption growth rate and time growth rate would directly correlate with the size of the input vectors defining the potential prime implicants of the Boolean function. Therefore, the authors expected the resource consumption growth rate of the implementation to be close to 3, since the number of potential prime implicants triples by adding a further input variable to the Boolean function and the response time growth rate to be much under 2. The time required to calculate the prime implicants using the particular modules was set using the minimum clock period allowing correct operation of the particular module. The authors also monitored the development of this characteristic in comparison with the input and output vector size.

As it has been stated in the previous sections, the module design was based on modules implemented as combinational logic circuits without any clock signal. Since specifying the minimum clock period using the Xilinx Vivado Design Suite HLx Edition 2016.2 tool requires using a flip-flop on both the input and the output side of the circuit, every top module had to be extended by a clock input. For testing purposes, the authors used the default circuit synthesis strategy, Vivado Synthesis Defaults 2016.

A summary of the implementation result may be found in the tables below. Figure 4 and Table 5 show the lookup table (LUT) and flip-flop consumption for the particular modules. The synthesis results confirmed a sub-linear increase in the number of consumed LUT resources, even though this was due to the increase of the AND/NAND gate input count (see also Figure 2), representing the prime implicants, the growth rate of the consumed LUT resources exceeds 3. As it is evident from Figure 4, there is a slight oscillation in the growth factor of the consumed LUT resources. The LUT resource consumption growth rate oscillation is caused by the Vivado synthesis tool, which uses an LUT-optimization technique to combine 3-input and 4-input LUTs to 5 and 6-input LUTs, implemented in Kintex-7 FPGA chips. Table 6 and the Figure 5 show a timing report of implemented modules. The authors focus their attention on the data path delay. The data path delay is the delay measured on the data path from the source to the destination. It indicates the module speed; in other words, it defines the response time. The results show that growth rate of the response time is much lower than 2, as was expected.

However, it is worth noting that the particular times define the minimum clock period on condition of implementing the computation for Boolean functions having an output from the set $\{0, 1\}$ solely; in this case, the computation time is the one stated in the table 5. Of the computation is implemented for Boolean functions with the output values from the set $\{0, 1, \times\}$, two computation steps have to be performed, as it has been stated in section 3 above and so the response time doubles.

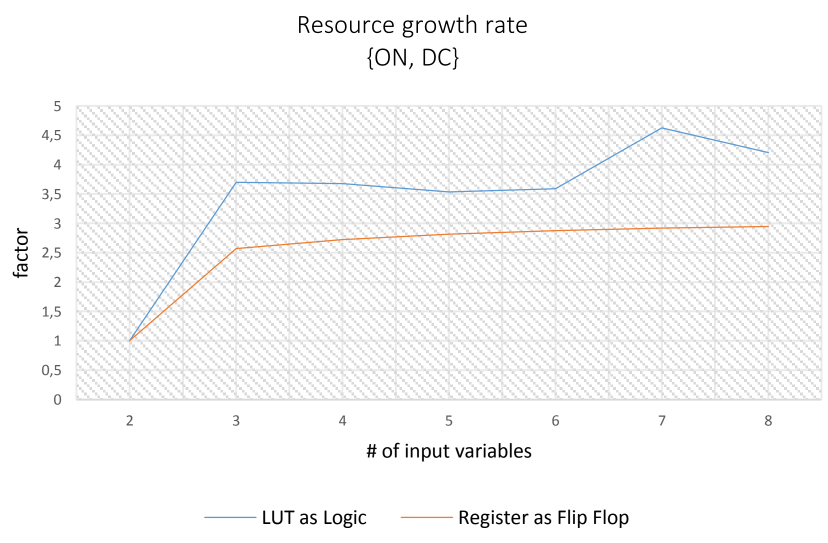

Figure 4

Resource consumption growth rate

Table 5

Summary of resource utilization and timing

Variable count	Resource Utilization		Timing Summary	
	LUT as Logic	Register as Flip Flop	Data Path Delay (Max Delay Path) [ns]	Logic Levels
2	20	28	2.818	2
3	74	72	4.396	3
4	272	196	7.77	6
5	962	552	11.336	8
6	3453	1588	14.328	10
7	15960	4632	21.88	27
8	67080	13636	30.052	39

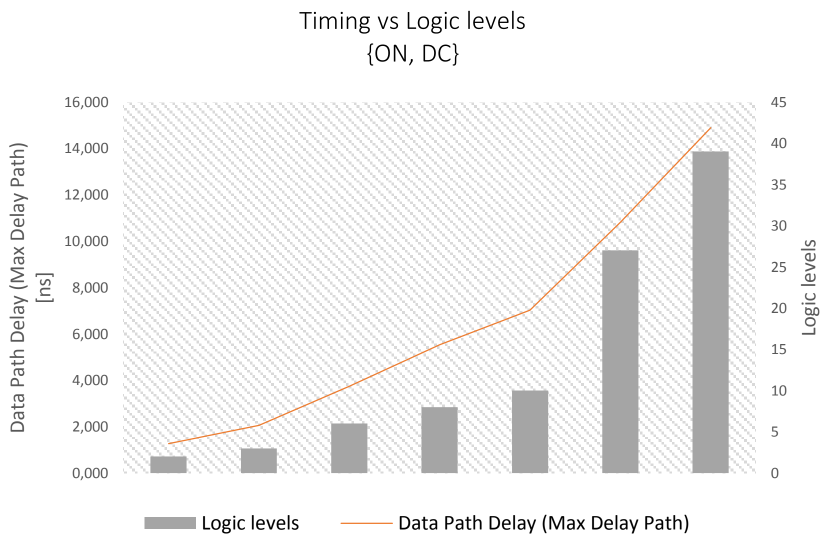

Figure 5
Timing summary

Conclusions

In this paper, the authors focused on the issue of accelerating Boolean function minimization. Systematic minimization, such as, the visual minimization method using Karnaugh map or the Quine-McCluskey algorithm implemented as a program, are two-step methods. The first step is the systematic generation of all prime implicants of a particular Boolean function. The second step is finding coverage of a particular Boolean function using the least possible prime implicants.

The work herein, is based on the previous development in this field [6], that proposed a combination logic circuit allowing the execution of the first step of systematic Boolean function minimization, i.e. allowing the generation of prime implicants of Boolean functions, on condition the particular function had fully defined output values. Defining DC output of the Boolean function was impossible. The solution proposed in this paper is an enhancement of the previous work, in which the possibility to select the mode of prime-implicant-generation for the DNF or CNF forms was added, along with the possibility to define DC outputs of the Boolean function. To generate prime implicants, the proposed hardware accelerator uses combinational logic; if the output values of the Boolean function belong only to the ON and OFF sets, it allows the generation of the prime implicant set of a particular Boolean function in a single step. If the output values of the particular Boolean function belong to the ON, OFF and DC sets, prime implicants will be generated in two steps. First, the prime implicants are generated, including those for which the Boolean function has only DC output values, i.e. they don't belong to the set of valid prime implicants of the particular Boolean function. In the second step, these invalid prime implicants are identified and then excluded from the set of prime implicants. The output of the proposed hardware accelerator is then a vector concerning valid prime implicants of the particular Boolean function.

The aim of this paper was to create a solution with exceptionally low time requirements, to generate the prime implicants of the particular Boolean function, which was achieved when the set of prime implicants could be generate for the tested Boolean functions in a matter of nanoseconds to tens of nanoseconds, the authors consider to be the main advantage of the proposed solution. Another advantage is that the time complexity depends only on the number of input variables of the Boolean functions and for the particular variable count, it is constant, regardless of the cardinality of the ON, OFF and DC sets. These advantages were achieved at the cost of spatial complexity of the proposed solution, thus, the implementation of the circuit is resource intensive, which is a disadvantage of the solution. The test of the proposed hardware accelerator, with its implementation for various amounts of input variables of the Boolean function, was performed using the Xilinx Kintex-7 FPGA KC705 Evaluation Kit evaluation board.

In future research, the authors shall focus on the possibility of decreasing the spatial complexity of the proposed solution, while maintaining the exceptionally low time requirements and allow the implementation of a further level of systematic minimization, i.e. the solution of Boolean function coverage using a FPGA hardware accelerator.

Acknowledgements

This research was supported by the Slovak Research and Development Agency, project number APVV-18-0214 and by KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 003TUKE-4/2017 Implementation of Modern Methods and Education Forms in the Area of Security of Information and Communication Technologies towards Requirements of Labor Market.

References

[1] A. Zhaparova, D. Titov, A. Y. Balkanov, G. Gyorok, G. "Study of the Effectiveness of Switching-on LED Illumination Devices and the Use of Low Voltage System in Lighting", Acta Polytechnica Hungarica, Óbuda University, Budapest, Hungary, Vol. 12, Issue 5, pp. 71-80, 2015, ISSN: 1785-8860

[2] A. Baláž and R. Hlinka, "Forensic analysis of compromised systems," 2012 IEEE 10th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stara Lesna, 2012, pp. 27-30

[3] V. Siládi and T. Filo, „Quine-McCluskey algorithm on GPGPU“, 3rd World Conference on Innovation and Computer Science (INSODE-2013) April 26-29, 2013, Antalya, Turkey, pp. 815-820, DOI: 10.13140/2.1.2113.1522

[4] V. Siládi, M. Povinsky, M. Povinsky, Ľ. Trajtel and M. Satymbekov, „Adapted parallel quine-McCluskey algorithm using GPGPU“, 2017 IEEE 14th International Scientific Conference on Informatics, November 14-16, 2017, Poprad, Slovakia, DOI: 10.1109/INFORMATICS.2017.8327269

[5] I. Savran and J. D. Bakos, „GPU Acceleration of Near-Minimal Logic Minimization“,Symposium on Application Accelerators in High Performance Computing, 2010

[6] B. Madoš, Z. Bilanová, E. Chovancová and N. Ádám, “Field Programmable Gate Array Hardware Accelerator of Prime Implicants Generation for Single-Output Boolean Functions Minimization”, ICETA 2019 – 17th International Conference on Emerging eLearning Technologies and Applications, November 21-22, 2019, The High Tatras, Slovakia

[7] Marquand, “XXXIII: On Logical Diagrams for n terms”, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 5. 12 (75), pp. 266-270, doi:10.1080/14786448108627104

[8] H. H. Aitken, “Synthesis of electronic computing and control circuits”, Harward University Press, Cambridge, Massachusetts, 1951, p. 294

[9] E. W. Veitch, “A Chart Method for Simplifying Truth Functions”, Proceedings of the 1952 ACM Annual Meeting (Pittsburgh, Pennsylvania, USA) New York, USA: Association for Computing Machinery (ACM), pp. 127-133, doi:10.1145/609784.609801

[10] M. Karnaugh, “The Map Method for Synthesis of Combinational Logic Circuits”, Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics. 72 (5), 1953, pp. 593-599, doi:10.1109/TCE.1953.6371932

[11] A. Svoboda, „Graficko-mechanické pomůcky užívané při analyse a synthese kontaktových obvodů” [Utilization of graphical-mechanical aids for the analysis and synthesis of contact circuits]. Stroje na zpracování informací [Symphosium IV on information processing machines] (in Czech) IV. Prague: Czechoslovak Academy of Sciences, Research Institute of Mathematical Machines. pp. 9-21

[12] M. V. Quine, “The Problem of Simplifying Truth Functions”, Amer. Math. Monthly, Vol. 59, 1952, No. 8, pp. 521-531

[13] E. J. McCluskey, “Minimization of Boolean functions”, The Bell System Technical Journal, 35, No. 5, Nov. 1956, pp. 1417-1444

[14] S. J. Hong, R. G. Cain and D. L. Ostapko, “MINI: A heuristic approach for logic minimization”, IBM Journal of Res. & Dev., Sept. 1974, pp. 443-458

[15] R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis" (9th printing 2000, 1st ed.). Kluwer Academic Publishers. ISBN 0-89838-164-9

[16] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis", Memorandum No. UCB/ERL M86-65. 5th June 1986, Berkeley, p. 140

[17] Espresso source code, University of California, Berkeley, https://ptolemy.berkeley.edu/projects/embedded/pubs/downloads/espresso/

[18] P. McGeer, J. V. Sanghavi, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions", Proc. DAC'93, 1996, pp. 432-440

[19] P. Fišer and J. Hlavička, BOOM, "A Heuristic Boolean Minimizer", Computing and Informatics. Vol. 22, pp. 19-51, 25 June 2003, ISSN: 2585-8807

[20] P. Fišer and J. Hlavička, „Efficient Minimization Method for Incompletely Defined Boolean Functions", Proceedings of the 4th International Workshop on Boolean Problems, University of Mining and Technology, Freiberg, Germany), IWSBP 4, September 21-22, 2000, pp. 91-98, ISBN: 3-86012-124-3

[21] J. Hlavička and P. Fišer, A Heuristic Method of Two-Level Logic Synthesis. Proceedings of The 5th World Multiconference on Systemics, Cybernetics and Informatics ISAS-SCI'2001, Orlando, Florida (USA), July, 22-25, 2001, Vol. XII, pp. 283-288, ISBN 980-07-7541-2

[22] P. Fišer and J. Hlavička, "On the Use of Mutations in Boolean Minimization", Proceedings of the Euromicro Symposium on Digital Systems Design, Warsaw, Sep. 4-6, 01, pp. 300-307

[23] J. Hlavička and P. Fišer, BOOM — a Heuristic Boolean Minimizer. Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001, IEEE Computer Society 2001, pp. 439-442, ISBN 0-7803-7249-2

[24] P. Fišer and H. Kubátová, "Boolean Minimizer FC-Min: Coverage Finding Process", Proc. 30th Euromicro Symposium on Digital Systems Design (DSD'04), Rennes, 31.8.-3.9.04, pp. 152-159

[25] P. Fišer and H. Kubátová, Two-Level Boolean Minimizer BOOM-II, Proc. 6th International Workshop on Boolean Problems (IWSBP'04), Freiberg, Germany, 23-24.9.2004, pp. 221-228

[26] P. Fišer and H. Kubátová, Flexible Two-Level Boolean Minimizer BOOM II and Its Applications, Proc. 9th Euromicro Conference on Digital Systems Design (DSD'06), Cavtat, (Croatia), 30.8.–1.9.2006, pp. 369-376