Minerva: Automated Hardware Optimization Tool

Farnoud Farahmand, Ahmed Ferozpuri, William Diehl and Kris Gaj
Department of Electrical and Computer Engineering, George Mason University
Fairfax, VA, U.S.A.
Email: {ffarahma, aferozpu, wdiehl, kgaj}@gmu.edu

***Abstract*—A common way of determining the maximum clock frequency of a digital system is static timing analysis provided by CAD toolsets, such as Xilinx Vivado, Xilinx ISE, and Intel Quartus Prime. Finding the actual maximum clock frequency is difficult, especially in Xilinx Vivado, due to the multitude of tool options, and a complex dependence between the requested clock frequency and the actual clock frequency achieved by the tool. For example, a binary search to find maximum frequency is tedious, time-consuming, and often does not obtain the correct result. In this research, we introduce an automated hardware optimization tool called Minerva. Minerva determines the close-to-optimal settings of tools, using static timing analysis and a heuristic algorithm developed by the authors, and targets either optimal throughput or throughput-to-area (TPA) ratio. We apply Minerva to the hardware benchmarking of authenticated cipher candidates competing in the CAESAR cryptographic contest, where best TPA ratio (without any specific target for maximum clock frequency) is one metric by which winners are selected. We evaluate RTL designs of 29 Round 2 CAESAR candidates and the current standard, AES-GCM, in terms of throughput and TPA ratio. Compared to a binary search for maximum frequency, our results demonstrate up to 25% improvement in terms of throughput, and up to 38% improvement in terms of TPA ratio.**

I. Introduction

Throughput, area, and throughput to area ratio are some of the most important metrics used for hardware evaluation. In hardware, the maximum throughput depends on the maximum clock frequency supported by each algorithm. The maximum clock frequency that can be achieved by a given RTL (Register-Transfer Level) code can be estimated or measured at different stages of the implementation process. The main stages are synthesis, placing and routing (P&R), and actual experimental testing on the board. The post-synthesis and post place & route results are determined by the FPGA tools using static timing analysis. There are two difficulties associated with static timing analysis of digital systems designed and modeled using hardware description languages, and implemented using FPGAs:

1) The latest version of CAD tools provided by Xilinx (Vivado), does not have the capability to report the maximum frequency achievable for the corresponding code. Essentially, the user requests a target frequency, and the tool reports either a "pass" or "fail" for its attempt to achieve this goal.

2) While there are 25 optimization strategies (i.e., sets of preselected option values) predefined in the tool, applying them sequentially, especially using the Graphical User Interface, is extremely tedious and time consuming.

Cryptographic contests have emerged as a commonly accepted way of developing cryptographic standards. This process has appeared to work very well in the case of Advanced Encryption Standard (AES), developed in the period 1997-2001 [1], and Secure Hash Algorithm 3 (SHA-3), developed in the period 2007-2012 [2]. At the same time, the observed increase in the number of algorithms qualified to the first round of the respective contests (51 in case of SHA-3 and 57 for CAESAR) inevitably brings the question of the efficiency of the current benchmarking approach. The number of candidates submitted to the first round of CAESAR (57) has exceeded the number of submissions to any previous contest, confirming the aforementioned trend. Similarly, the numbers of candidates qualified to the second rounds of their respective competitions have increased from 5 in the case of AES, through 14 for SHA-3, to 29 in the case of CAESAR. This issue also applies to post-quantum cryptography and the corresponding algorithms which are significantly more complex and harder to evaluate compared to authenticated ciphers and hash functions.

To overcome the aforementioned difficulties and facilitate hardware benchmarking of algorithms by static timing analysis methods, we introduce Minerva. Minerva is an automated and comprehensive hardware optimization tool. Minerva employs a unique heuristic algorithm, which is customized for frequency search using CAD toolsets, in addition to supporting other standard search techniques. It can incorporate an arbitrary number of predefined or user-defined strategies to achieve the highest possible frequency or frequency/area for each design. Moreover, it takes advantage of multithreading and multi-core execution to significantly reduce run time.

The use of an optimization tool, such as Minerva, is highly desirable for cryptographic contests, which determine relative efficiency based on the TPA ratio, e.g., Mbps/LUT or Mbps/slice for implementations in Xilinx FPGAs. In this paper, we report the Minerva optimized results in terms of Throughput, Area and Throughput to Area ratio for the RTL VHDL code of 29 Round 2 CAESAR candidates and AES-GCM [3]. Results are separately reported for all three optimization modes supported by Minerva. We then compare Minerva results with the results generated using a traditional binary search in Xilinx Vivado. Additionally, the run times of both methods (i.e., the three Minerva modes and the binary search) are reported for all of these authenticated ciphers.

This work is supported by NSF Grant #1314540

978-1-5386-3797-5/17/$31.00 ©2017 IEEE

II. Previous Work

A tool called SUPERCOP, which expedites comparison of software implementations of cryptographic algorithms, is presented in [4]. This open source tool supports the choice of the best compilation options from thousands of different combinations. It also facilitates execution time measurements on multiple computer systems.

In [5], an open-source environment for fair, comprehensive, automated, and collaborative hardware benchmarking of algorithms belonging to the same class is presented. The main part of this environment is the ATHENa tool for optimization of tool options, requested clock frequency, and the starting point of placement. ATHENa provides capabilities similar to our Minerva capabilities for designers targeting FPGA devices from two major vendors, Xilinx and Altera. However, it works only with the previous-generation Xilinx CAD tool (ISE), which will not support Xilinx FPGAs beyond the Series 7 families (Virtex-7, Kintex-7, Artix-7).

Moreover, FPGA vendors themselves have their own tools for the exploration of implementation options. One example is ExploreAhead [6] from Xilinx, which is a part of the high-level optimization tool called PlanAhead. PlanAhead is provided as a built-in option in Vivado Design Suite, the latest version of Xilinx CAD tools. ExploreAhead allows executing multiple implementation runs based on predefined or user-defined strategies (understood as preselected values for a set of options). Additionally, it supports parallel runs on multi-core CPUs. Unlike ATHENa, which supports two vendors, PlanAhead works only with Xilinx FPGAs. Additionally, ATHENa is aimed at achieving the best possible performance (e.g., the best throughput/area ratio), while ExploreAhead and Vivado aim only at achieving the requested clock frequency.

In [7], the authors present InTime, a machine learning approach, supported by a cloud-based compilation infrastructure, to automate the selection of FPGA CAD tool parameters and minimize the TNS (total negative slack) of the design. A combination of open-source and industrial benchmarks that occupy between 50-90% of the FPGA capacity have been investigated to measure the efficiency and capability of this tool. The results demonstrate up to 70% timing improvement on modern Altera FPGAs. However, InTime is a commercial tool, which may be too expensive for use in academia and in small companies. On the other hand, Minerva is a free and open-source tool, and its source code and user's manual are available at [8]. In addition, InTime does not have the capability to find the actual maximum frequency with positive TNS near zero; it just tries to find the best tool options to minimize the WNS (Worst Negative Slack) corresponding to a specific design and user-defined timing constraints.

Experimental testing using actual hardware is an alternative method for hardware evaluation of maximum frequency. In [9], a Zynq-based testbed for hardware evaluation of cryptographic algorithms is reported. The authors measured the maximum frequency and throughput supported by 12 Round 2 SHA-3 candidates using two methods, experimentally, and using static timing analysis, and compared the results. In these results, the experimental maximum frequency was always higher than frequency achieved by static timing analysis, but the ratio of these two frequencies was a strong function of the implemented algorithm.

III. Environment

In order to observe the behavior of the Vivado Design Suite in static timing analysis, synthesis and implementation were performed for the VHDL code of 5 CAESAR Round 2 candidates [3]. At first, the same requested clock frequency constraint was used for each algorithm. The target clock frequency was set to 333 MHz, and the theoretically achievable frequency (further referred to as the reference frequency) was calculated based on WNS, utilizing the following formula:

$$Minimum\ Clock\ Period = Target\ Clock\ Period - WNS \quad (1)$$

In the next step, WNS results were generated for the requested clock frequency varying in range of -64 to +64 MHz of the reference frequency, with a precision of 1 MHz. In other words, the authors generated WNS results for 128 different target clock frequencies in order to observe a trend. Fig. 1, Fig. 2 and Fig. 3 show this trend for AES-GCM, SCREAM and ICEPOLE, respectively. The GraphGen function provided by Minerva accommodated the aforementioned process.

As observed in Fig. 2 and Fig. 3, there are fluctuations around the calculated reference clock frequency. This fluctuation is much higher in case of ICEPOLE. As a result, it would be very hard to find the actual maximum clock frequency without automation. In contrast, there are fewer fluctuations for AES-GCM. Based on Xilinx documentation [10], the only acceptable target frequency is the one that gives us positive slack. Therefore, based on the aforementioned graphs, we cannot rely on (1) to calculate the actual maximum clock frequency. Instead, we need a more complex procedure. In addition, these results are generated using only default options of Vivado for all implementation steps, such as mapping, placing and routing. The Vivado Design Suite ships with 25 predefined optimization strategies, which can be used to achieve a higher maximum frequency and a more optimized design. Hence, incorporating all of these strategies leads to an even more tedious process.

One way to find the maximum frequency in a given frequency range is to use a binary search algorithm. However, there are two problems associated with this method: 1) We cannot easily cover 25 optimization strategies, and 2) Based on the fluctuations observed in the generated graphs, different results will be achieved for different input ranges. Also, it is possible that none of the results will be the actual maximum clock frequency.

Fig. 3 indicates how the binary search scheme works to find the maximum achievable clock frequency between the graph generation input ranges. At first we check the lower bound and upper bound (number 1 and number 2 in the figure) to make sure we search in a correct range. In other

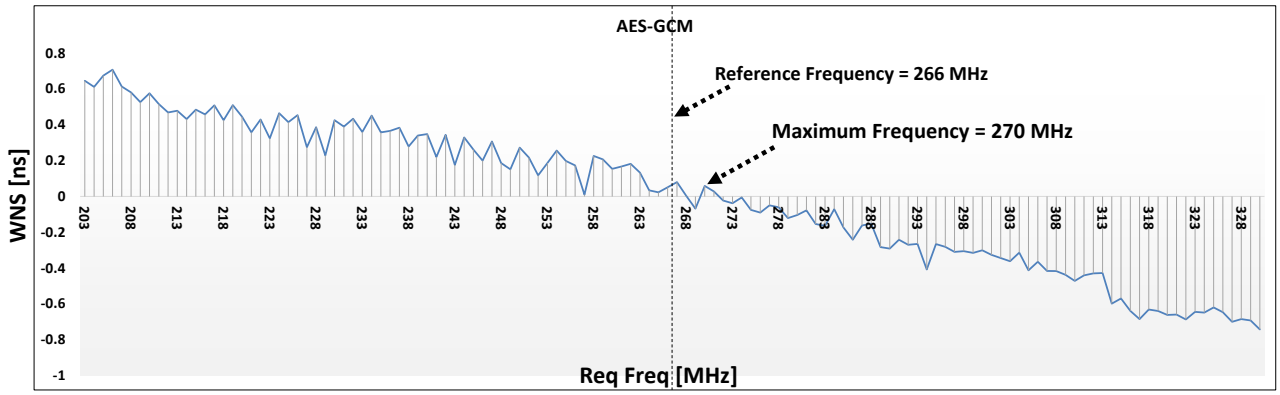


Fig. 1: Dependence of the Worst Negative Slack (WNS) on the Requested Clock Frequency (Req Freq) for the high-speed implementation of AES-GCM.

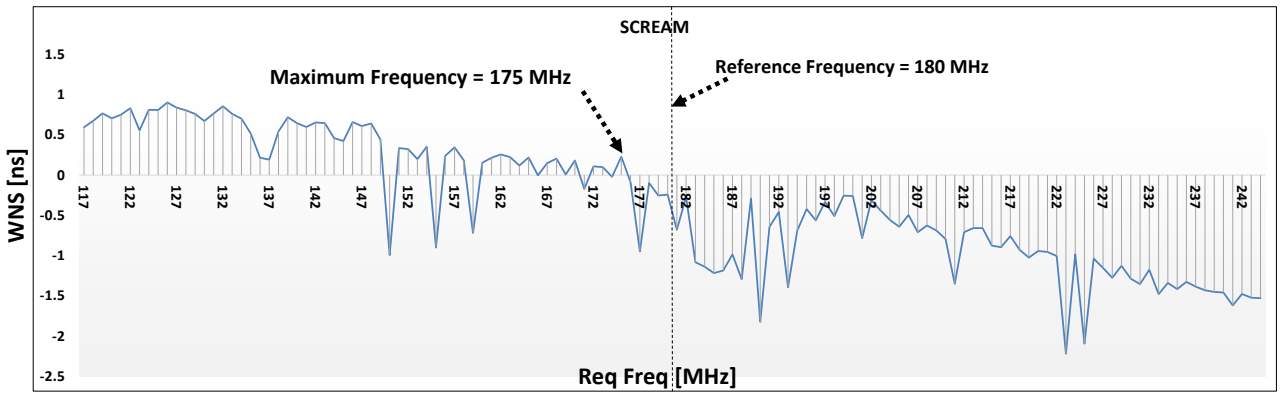


Fig. 2: Dependence of the Worst Negative Slack (WNS) on the Requested Clock Frequency (Req Freq) for the high-speed implementation of SCREAM.

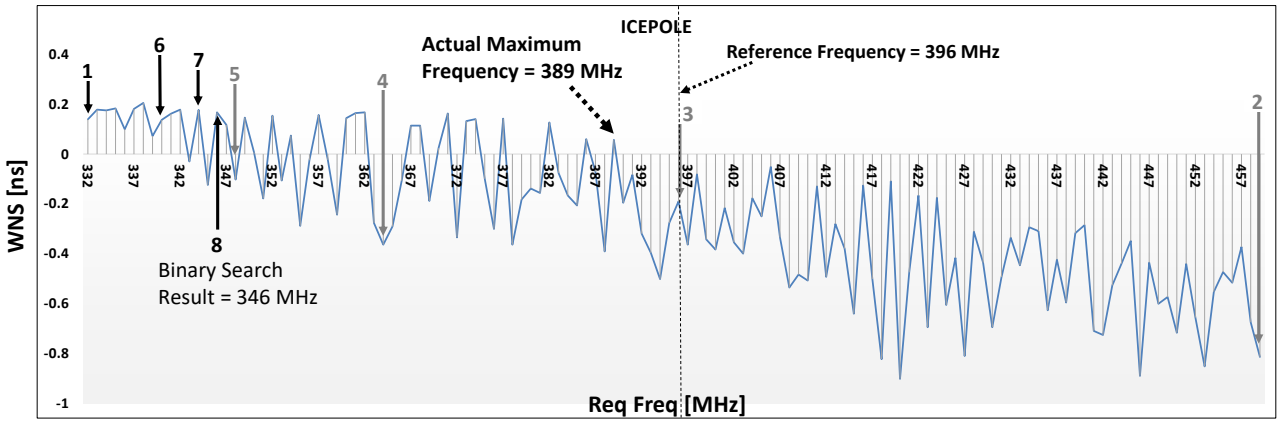


Fig. 3: Dependence of the Worst Negative Slack (WNS) on the Requested Clock Frequency (Req Freq) for the high-speed implementation of ICEPOLE, and the graphical representation of the binary search scheme.

words, we receive positive WNS for lower bound and negative WNS for upper bound frequencies; otherwise the input range should be updated. Then, we find the middle point of the aforementioned range (number 3 in the figure) and generate the timing result for that frequency. If the resultant WNS is positive, we will update the lower bound frequency with the middle point. Otherwise, the upper bound frequency should be reduced to the middle frequency. The aforementioned binary search scheme continues until we reach a precision of 1 MHz. As we can observe in Fig. 3, the binary search result in case of ICEPOLE is 346 MHz (number 8 in the figure), which is not the correct maximum frequency. Based on the ICEPOLE

graph, the maximum frequency is 389 MHz. As a result, we equip Minerva with a heuristic algorithm aimed at addressing this problem.

Minerva is used to execute Vivado in batch mode, utilizing the Vivado batch mode Tcl scripts provided by Xilinx. An XML-based Python program is used to manage runs. This program launches Vivado with Tcl scripts that are dynamically created during run-time and later modified to perform each step of the optimization algorithm. Minerva is designed to be used to automate the task of finding optimized results for each directory of a source code repository, and works with any device that Vivado supports.

IV. Design Flow

Minerva supports multiple frequency search algorithms, and supports addition of new algorithms in the future. In this work we implement three modes of Minerva frequency searches. The first mode (*Minerva_TP_Opt*) is designed specifically to find the maximum frequency achievable by a given hardware design. *Minerva_TP_Opt* function receives the following parameters as input:

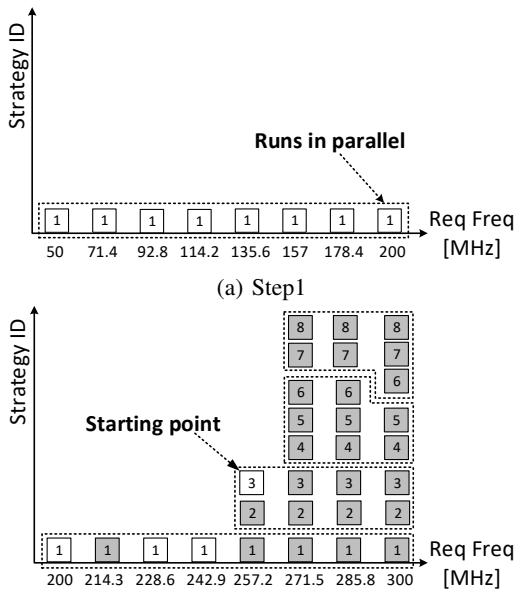
- fmin and fmax: these are the lower and upper bounds of the frequency range that we span to find the maximum frequency. These values can be updated during run-time.
- n: indicates the number of runs to be performed in parallel. Minerva can run on multiple CPU cores and take advantage of multithreading.
- p: represents the number of optimization strategies to be considered during the search.
- r (precision range size): is the maximum number of frequency targets (higher than the last achieved maximum clock frequency) to be explored. If we achieve positive slack for a frequency in this range, we will continue the search; otherwise we will terminate the process.

This function generates an output report that contains the following information:

1) WNS result for all test cases with the corresponding optimization strategy ID and target clock frequency.
2) WNS and Area results for all target frequencies with positive slack.
3) Maximum frequency with WNS $\geq$ 0, f_pass_max
4) Minimum Area in the number of LUTs achievable for f_pass_max (denoted by min_LUTs(f_pass_max)), the corresponding ratio f_pass_max/min_LUTs(f_pass_max), and the corresponding optimization strategy ID.
5) Minimum Area in the number of Slices achievable for f_pass_max (denoted by min_Slices(f_pass_max)), the corresponding ratio f_pass_max/min_Slices(f_pass_max), and the corresponding optimization strategy ID.
6) Execution time.

Please note that the Strategy IDs may be different for the outputs 4) and 5).

Fig. 4 (a)-(f) completely describes how *Minerva_TP_Opt* algorithm works. This figure is drawn assuming the following values of the Minerva parameters: fmin=50, fmax=200, n=8, r=8, and p=8. Each column illustrates one requested clock frequency value, and square blocks in that column correspond to optimization strategies. Each square block represents one test case with the optimization strategy ID mentioned inside it. Colors of these blocks are white or gray, indicating positive or negative WNS, respectively. The runs that execute in parallel at each step are represented using dotted boxes.



(a) Step1

(b) Step2

Fig. 4: Graphical representation of the Minerva frequency search algorithm Minerva_TP_Opt, with the parameters n=p=r=8. White and grey blocks indicate positive and negative WNS respectively.

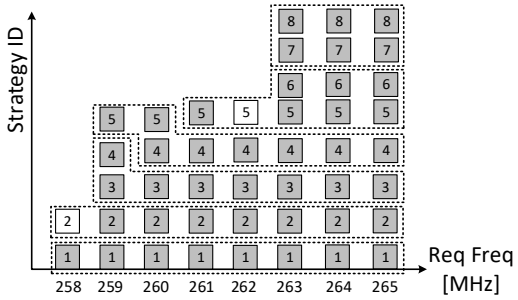
Fig. 4(a) shows the first step in *Minerva_TP_Opt* algorithm. In the first step, the given frequency range (50 to 200) is divided by $r-1$ to have 8 frequencies including 50 and 200, with the same distance between each other, as shown in Fig. 4(a) Freq axis. Then, WNS results are generated for all of these 8 target frequencies and the default optimization strategy. It is feasible to run all of these target frequencies at the same time, as n is equal to 8 in this example. After WNS results are generated, if the upper bound frequency (fmax) gives us positive slack, we update fmin and fmax values using (2) and (3), and repeat the previous process (step forward).

$$fmin(new) = fmax(old) \tag{2}$$

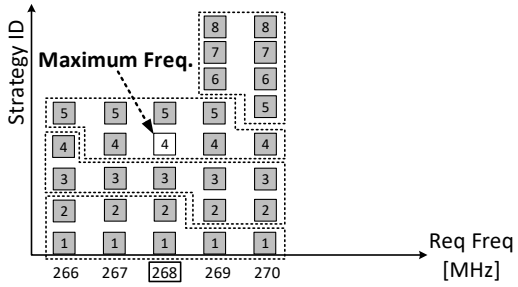
$$fmax(new) = fmax(old) + 100 \tag{3}$$

If all of the first 8 target clock frequencies give us negative slack, we step backward by a frequency range of 100 MHz. Accordingly, fmin and fmax are updated using (4) and (5), and the first step is repeated.

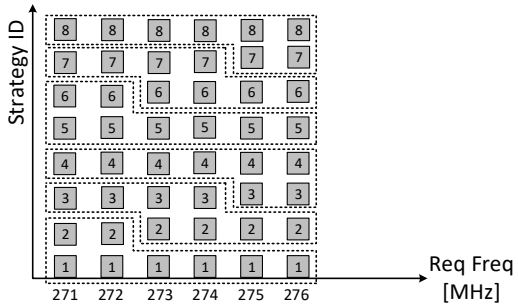
$$fmin(new) = fmin(old) - 100 \tag{4}$$



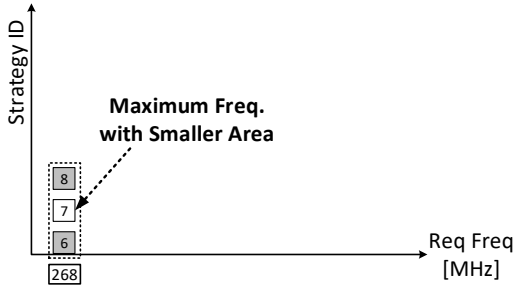
(c) Step3



(d) Step4



(e) Step5



(f) Step6

Fig. 4: Graphical representation of the Minerva frequency search algorithm Minerva_TP_Opt, with the parameters n=p=r=8. White and grey blocks indicate positive and negative WNS respectively.

$$fmax(new) = fmin(old) \tag{5}$$

The aforementioned process leads to finding the maximum frequency, less than fmax, that gives us positive slack using only the default optimization strategy. As we can observe in Fig. 4(a), in the first step, positive slack is achieved for fmax (200 MHz). Hence, we step forward and update fmin and fmax to 200 and 300 MHz respectively, see Fig. 4(b). As shown in this figure, 242.9 MHz is the highest frequency that leads to positive slack with the default optimization strategy.

At this point, the optimization runs are started for the remaining frequencies in this range higher than 242.9 MHz. In this example 257.2 MHz, with optimization strategy number 3 has positive slack, so the maximum frequency is updated to 257.2 MHz. In case of higher frequencies, all 8 optimization strategies fail. Therefore, 257.2 MHz becomes our starting point to begin the next step of frequency search considering 8 optimization strategies and a precision of 1 MHz.

The next step is illustrated in Fig. 4(c). In this step we go forward by 1 MHz. As soon as we find a frequency with positive slack, the lower frequencies and the remaining optimization strategies corresponding to these frequencies are eliminated. The aforementioned procedure is continued until 8 (precision range size) consecutive frequencies fail to provide positive slack for all possible optimization strategies (8 in this example), as shown in Fig. 4(d) and Fig. 4(e). Therefore, in this example, the maximum frequency with WNS $\geq$ 0, f_pass_max, is 268 MHz, using the optimization strategy number 4.

Let us assume that the number of LUTs for Strategy 4 is 1000, and the number of Slices 300. Based on Fig. 4(d), only the first 5 optimization strategies were tested for f_pass_max=268 MHz. Therefore, in the next step, shown in Fig. 4(f), we perform runs for the remaining three strategies at the same maximum clock frequency of 268 MHz. As we can see in this figure, only one of these runs passes with WNS $\geq$ 0, for the strategy ID=7. Now let us assume that the corresponding areas for Strategy 7 are 970 LUTs and 310 Slices. Then, the algorithm returns two sets: {f_pass_max=268 MHz, Minimum number of LUTs achievable for f_pass_max, min_LUTs(268 MHz)=970, the corresponding ratio f_pass_max/min_LUTs(f_pass_max)=268/970, and the corresponding optimization strategy ID=7} as well as {f_pass_max=268 MHz, Minimum number of Slices achievable for f_pass_max, min_Slices(268 MHz)=300, the corresponding ratio f_pass_max/min_Slices(f_pass_max)=268/300, and the corresponding optimization strategy ID=4}.

The second mode of Minerva frequency search (*Minerva_TPA_Opt*) targets further optimization of the frequency to #LUTs ratio (Throughput to area ratio). This mode can be used after *Minerva_TP_Opt* search generates the maximum frequency. Minerva_TPA_Opt receives the following parameters as input: 1) f_pass_max (maximum frequency achieved by *Minerva_TP_Opt mode*), 2) n (number of runs in parallel) and 3) p (number of optimization strategies). The output report contains the same information as the first mode (*Minerva_TP_Opt*). In this mode, we generate the results for all the frequencies between 96% of f_pass_max and f_pass_max, with a precision of 1 MHz. We also try all possible optimization strategies. At the end, the requested frequency and optimization strategy combination that leads to the best TPA is reported.

The third mode of Minerva frequency search (*Minerva_Fast_Opt*) is designed to achieve proper results in terms

TABLE I: Detailed values of the maximum clock frequency (MHz), area (number of LUTs) and frequency/LUT generated using three modes of Minerva and binary search for 29 Round 2 CAESAR candidates and AES-GCM

Algorithm	Minerva_TP_Opt			Minerva_TPA_Opt			Minerva_Fast_Opt			Binary search		
	Freq. [MHz]	#LUTs	Freq./ #LUTs	Freq. [MHz]	#LUTs	Freq./ #LUTs	Freq. [MHz]	#LUTs	Freq./ #LUTs	Freq. [MHz]	#LUTs	Freq./ #LUTs
ACORN	394	1,632	0.241	394	1632	0.241	345	1,475	0.234	374.32	1,626	0.230
AEGIS	368	7,504	0.049	368	7,504	0.049	356	7,509	0.047	329.49	7,484	0.044
AES-COPA	280	7,717	0.036	280	7,717	0.036	271	7,687	0.035	269.73	7,702	0.035
AEZ	394	5,167	0.076	394	5,167	0.076	381	5,008	0.076	314.55	5,101	0.062
Ascon	451	1,564	0.288	451	1,564	0.288	442	1,542	0.287	427.05	1,542	0.277
CLOC	251	3,844	0.065	248	3,676	0.067	248	3,676	0.067	246.88	3,816	0.065
COLM	263	8,131	0.032	263	8,131	0.032	255	8,118	0.031	249.00	8,093	0.031
Deoxys	366	3,343	0.109	362	3,284	0.110	362	3,284	0.110	356.08	3,313	0.107
HS1-SIV	234	8,078	0.029	234	8,078	0.029	231	8,074	0.029	226.00	8,066	0.028
ICEPOLE	441	5,128	0.086	441	5,128	0.086	441	5,128	0.086	356.74	5,677	0.063
JAMBU-AES	288	1,696	0.170	285	1,595	0.179	285	1,595	0.179	273.24	1,688	0.162
Joltik	439	1,606	0.273	427	1,558	0.274	427	1,558	0.274	413.87	1,592	0.260
KetjeJr	269	1,320	0.204	259	1,200	0.216	259	1,200	0.216	253.96	1,306	0.194
Minalpher	242	7,947	0.030	234	7,381	0.032	234	7,381	0.032	202.93	7,326	0.028
MORUS	299	4,594	0.065	299	4,594	0.065	295	4,615	0.064	283.79	4,586	0.062
NORX	206	4,474	0.046	201	3,665	0.055	201	3,665	0.055	197.88	4,371	0.045
OCB	351	4,483	0.078	351	4,483	0.078	342	4,463	0.077	334.82	4,487	0.075
OMD	301	4,624	0.065	301	4,624	0.065	282	4,618	0.061	259.18	4,586	0.057
PAEQ	322	8,373	0.038	322	8,373	0.038	294	8,331	0.035	280.27	8,334	0.034
π-Cipher	209	3,861	0.054	209	3,861	0.054	201	3,843	0.052	192.38	3,838	0.050
POET	249	7,487	0.033	249	7,487	0.033	249	7,487	0.033	216.11	7,380	0.029
PRIMATEs-GIBBON	216	1,942	0.111	216	1,942	0.111	202	1,886	0.107	185.35	1,827	0.101
PRIMATEs-HANUMAN	215	1,891	0.114	215	1,891	0.114	198	1,805	0.110	195.02	1,796	0.109
RiverKeyak	192	8,169	0.024	192	8,169	0.024	172	7,823	0.022	160.74	7,699	0.021
SCREAM	187	2,614	0.072	187	2,614	0.072	187	2,614	0.072	176.78	2,764	0.064
SILC	347	3,110	0.112	341	3,039	0.112	312	2,952	0.106	306.70	3,062	0.100
STRIBOB	381	4,670	0.082	381	4,670	0.082	370	4,648	0.080	369.98	4,648	0.080
Tiaoxin	296	7,556	0.039	296	7,556	0.039	280	7,539	0.037	258.30	7,492	0.034
TriviA-ck	255	2,587	0.099	255	2,587	0.099	242	2,573	0.094	233.25	2,571	0.091
AES-GCM	277	3,105	0.089	277	3,105	0.089	271	3,089	0.088	271.92	3,097	0.088

of both throughput and throughput to area ratio in a short amount of time compared to the first and second modes. Based on the results generated for 30 benchmarked authenticated ciphers, we arrived at the optimization strategy that gave us the best throughput to area ratio in most cases, and utilized it as a single optimization strategy. This optimization strategy focused on reducing area by ExploreArea command. Therefore, *Minerva_Fast_Opt* works similar to *Minerva_TP_Opt*; the only difference is the number of optimization strategies, i.e., two optimization strategies in case of *Minerva_Fast_Opt*, namely, the default one and the one based on the ExploreArea command.

V. Results

Vivado Design Suite 2015.1 is used for result generation. The target device is set to the Virtex-7 (xc7vx485-tffg1761-3). Binary search is done by considering only the default optimization strategy, and Minerva frequency search is configured using the following values: n = 16, p = 23, r = 12, and the input range is [100, 500] for all candidates.

Table I presents detailed values of the performance metrics generated using the three modes of Minerva frequency search and binary search for the VHDL code of 29 Round 2 CAESAR candidates and AES-GCM [3]. For each mode, the first and second columns show frequency in MHz and area in the number of LUTs, respectively, obtained by utilizing a Minerva frequency search in the corresponding mode, or binary search. The third column reports the ratio of frequency to area (in number of LUTs) calculated based on the results in the first and second columns. The first, second and third set of results are generated by *Minerva_TP_Opt*, *Minerva_TPA_Opt* and *Minerva_Fast_Opt* modes of operation, respectively, and the final set of results is acquired using binary search with the default optimization strategy.

Fig. 5 presents the ratio of results obtained using the three modes of Minerva frequency search vs. Binary search in terms of Throughput. *Minerva_TP_Opt* is always guaranteed to return the best Throughput compared to the remaining two modes. *Minerva_TPA_Opt* is usually the second best, due to the different optimization target. *Minerva_Fast_Opt*, as expected, is somewhat lagging behind, but it still outperforms binary search for 28 out of 30 algorithms, reaching in 3 cases the same performance as *Minerva_TP_Opt*, and in 10 cases the same performance as *Minerva_TPA_Opt*.

Fig. 6 illustrates the ratio of results obtained using the three modes of Minerva frequency search vs. Binary search in terms of TPA. The order of candidates is based on the decreasing improvement of *Minerva_TPA_Opt* over Binary search. Our results show that the TPA ratio has improved by almost 38% for ICEPOLE, and more than 20% in case of AEZ and NORX. This metric has improved by more than 15% in case of OMD, and by more than 10% for the next 10 candidates.

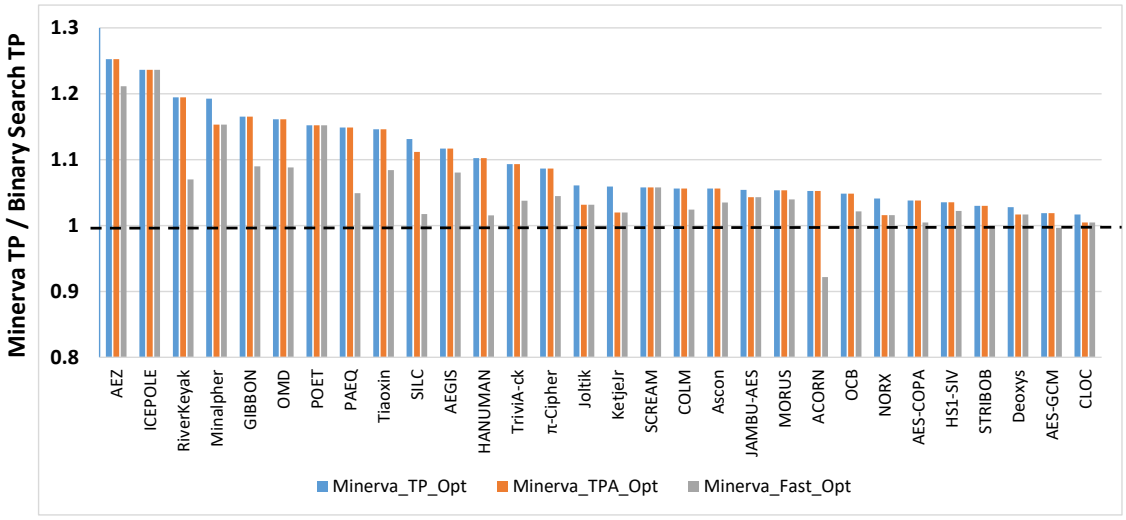


Fig. 5: Ratios of Minerva TP / Binary Search TP for three modes of Minerva frequency search, and 30 authenticated ciphers. Notation: TP - Throughput.

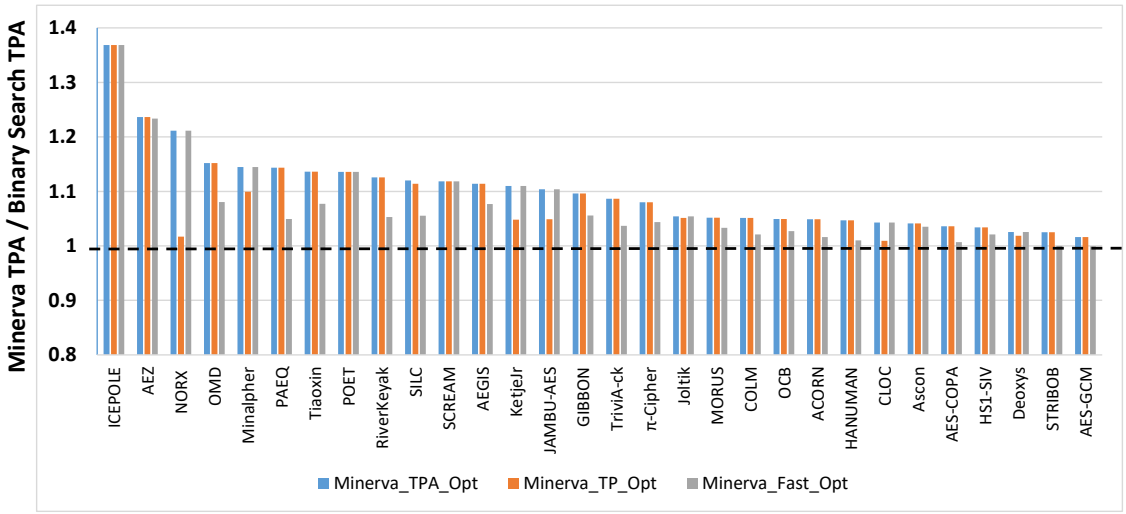


Fig. 6: Ratios of Minerva TPA / Binary Search TPA for three modes of Minerva frequency search, and 30 authenticated ciphers. Notation: TPA - Throughput/Area ratio.

As expected, algorithms which have more fluctuations around the reference frequency in the previously generated graphs, such as ICEPOLE (Fig. 3), take better advantage of Minerva frequency searches than the stable ones, such as AES-GCM (Fig. 1) (i.e., 38% vs. less than 5%).

Minerva_Fast_Opt gives the same TPA as *Minerva_TPA_Opt* for 10 algorithms. Somewhat surprisingly, *Minerva_TP_Opt* gives worse performance than *Minerva_Fast_Opt* for 7 authenticated ciphers, e.g., NORX, despite the longer execution time. This behavior is caused by the fact that the best TPA is achieved for a frequency different than f_pass_max, and only the best TPA ratios corresponding to f_pass_max are returned by *Minerva_TP_Opt*.

The computer system used for the optimization runs has the following specification: Intel Xeon CPU E5-2667 v3, 3.20GHz, 32 CPUs, 128 GB RAM, Ubuntu 14.04 LTS. Table II presents the execution times for the three modes of Minerva frequency search and the binary search, respectively. As shown in this table, similarly to the TPA ratio improvement, *Minerva_TP_Opt* run time depends on the corresponding candidate's graph stability. AES-GCM, the algorithm with the most stable graph, has the lowest run time (3 hours and 20 minutes) and ICEPOLE, for which the graph shows the most fluctuations, has one of the highest execution times (12 hours and 41 minutes). In addition, the Minerva run time has a direct relation with n (number of runs in parallel) which is 16 in this case. On the other hand, the times of the binary searches are very consistent for all 30 algorithms. In addition, as presented in Table II, *Minerva_Fast_Opt* has a much lower run time

TABLE II: Run time for binary search and three modes of Minerva frequency search for 29 Round 2 CAESAR candidate and AES-GCM

Algorithm	Run time [hrs:min]			
	Minerva TP_Opt	**Minerva TPA_Opt**	**Minerva Fast_Opt**	**Binary search**
ACORN	6:13	9:43	1:25	0:44
AEGIS	7:49	12:19	2:02	0:40
AES-COPA	7:57	10:57	2:00	1:00
AEZ	6:18	9:52	1:00	1:00
Ascon	3:15	6:41	0:56	0:50
CLOC	5:46	7:48	1:25	1:35
COLM	6:57	9:31	1:02	0:50
Deoxys	3:14	6:25	1:16	1:04
HS1-SIV	5:14	7:11	1:05	1:10
ICEPOLE	12:41	18:27	2:13	1:00
JAMBU-AES	4:45	7:27	0:38	0:40
Joltik	5:24	8:21	0:56	0:45
KetjeJr	4:01	6:06	1:14	0:51
Minalpher	8:23	11:18	1:40	1:00
MORUS	6:50	9:27	2:30	0:54
NORX	6:09	8:14	0:49	1:15
OCB	5:11	8:11	1:58	1:20
OMD	6:01	8:21	0:55	0:50
PAEQ	13:42	17:59	4:18	1:04
π-Cipher	5:21	7:11	0:56	0:50
POET	6:44	9:05	1:42	1:00
GIBBON	5:31	7:23	1:19	1:00
HANUMAN	5:30	7:18	0:35	1:00
RiverKeyak	12:00	16:55	3:20	0:59
SCREAM	6:52	8:36	1:17	1:10
SILC	7:02	9:30	0:41	0:55
STRIBOB	5:12	8:45	3:45	1:30
Tiaoxin	11:05	14:18	1:57	0:45
TriviA-ck	5:47	7:56	0:46	1:04
AES-GCM	3:34	5:26	0:39	1:04
Average Run time	**6:40**	**9:33**	**1:32**	**0:59**

compared to other two modes, and is even faster than a binary search in case of 7 algorithms.

VI. Conclusions

We have introduced an automated hardware optimization tool called Minerva, and demonstrated its utility toward achieving optimal performance during benchmarking of a large number of RTL designs of authenticated ciphers. Minerva searches for the best requested clock frequency and the best set of tool options, leading to the highest achieved clock frequency, or the highest achieved frequency to area ratio, after static timing analysis. In addition, Minerva takes advantage of multithreading and multi-core execution to reduce run time. It can apply an arbitrary number of preselected tool option sets (called optimization strategies), and combine them with a frequency search in order to achieve the best results in terms of throughput, or throughput to area ratio. The results for 29 Round 2 CAESAR candidates and AES-GCM indicate that we can achieve up to 38% improvement in terms of the throughput to area ratio in comparison to a simpler binary search for the optimal requested clock frequency, using default values of all tool options. The average run time depends mostly on n (number of runs in parallel) which was 16 in our experiments. This average run time is over 6 and 9 times longer than the run times for binary searches in case of *Minerva_TP_Opt*, and *Minerva_TPA_Opt* modes, respectively. However, the third mode of Minerva (*Minerva_Fast_Opt*) has an execution time tantamount to a binary search, and produces acceptable results, compared to the other two modes of Minerva.

Therefore, the choice of operation mode depends on the user expectation. *Minerva_TP_Opt* provides the maximum frequency in a moderate amount of time. *Minerva_TPA_Opt*, which runs on top of *Minerva_TP_Opt*, produces the best results in terms of throughput/area, but takes more time to execute. Finally, *Minerva_Fast_Opt* produces fair results in terms of both throughput and throughput/area in a very short amount time - sometimes even faster than a binary search.

Our future work will involve attempts at further run time optimization to reduce Minerva execution times by using methods such as machine learning algorithms. In addition, *Minerva_Fast_Opt* can be enhanced with additional customized optimization strategies to generate improved results in a short amount of time. Furthermore, we will be able to add support for Intel Quartus Prime and ASIC CAD tools. Finally, we should investigate the properties of authenticated ciphers that lead to good graph stability (i.e., low change in positive or negative slack around an optimal point of inflection), or poor graph stability, which can significantly affect run times of optimization tools.

References

[1] National Institute of Standards and Technology. (2000, Oct) Report on the development of the Advanced Encryption Standard (AES). [Online]. Available: http://csrc.nist.gov/archive/aes/round2/r2report.pdf

[2] ——. (2012, Nov) Third-round report of the SHA-3 cryptographic hash algorithm competition. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf

[3] GMU Source Code of Round 3 & Round 2 CAESAR Candidates, AES-GCM, AES, AES-HLS, and Keccak Permutation F. Accessed August 8, 2017. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=CAESAR_source_codes

[4] D. J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. Accessed August 1, 2017. [Online]. Available: http://bench.cr.yp.to

[5] K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster, "ATHENa - automated tool for hardware evaluation: Toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs," in *20th International Conference on Field Programmable Logic and Applications, FPL 2010*, Milano, Italy, Aug. 31st - Sep. 2nd, 2010, pp. 414–421.

[6] M. Goosman, R. Shortt, D. Knol, and B. Jackson, "ExploreAhead extends the PlanAhead performance advantage," in *Xcell Journal*, Third Quarter 2006, pp. 62–64.

[7] N. Kapre, H. Ng, K. Teo, and J. Naude, "InTime: A machine learning approach for efficient selection of FPGA CAD tool parameters," in *23rd ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2015*, Monterey, California, USA, Feb. 22-24, 2015, pp. 23–26.

[8] Minerva: Automated Hardware Optimization Tool. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=Minerva

[9] F. Farahmand, E. Homsirikamol, and K. Gaj, "A Zynq-based testbed for the experimental benchmarking of algorithms competing in cryptographic contests," in *2016 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2016*, Nov 2016, pp. 1–7.

[10] Xilinx. Vivado Design Suite User Guide. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug973-vivado-release-notes-install-license.pdf