2020•2021

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Embedded FPGA for cryptography

PROMOTOR :

Prof. dr. ir. Nele MENTENS

PROMOTOR :

Assistant Prof. Francesco REGAZZONI

Jelle Biesmans, Matthias Dekeyser

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Gezamenlijke opleiding UHasselt en KU Leuven

UHASSELT KU LEUVEN

UHASSELT KU LEUVEN

2020•2021

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Embedded FPGA for cryptography

PROMOTOR :

Prof. dr. ir. Nele MENTENS

PROMOTOR :

Assistant Prof. Francesco REGAZZONI

Jelle Biesmans, Matthias Dekeyser

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

UHASSELT KU LEUVEN

Preface

This master's thesis is a continuation of the research of Prof. Dr. Ir. Nele Mentens, Dr. Francesco Regazzoni and Prof. Dr. Edoardo Charbon. Their research introduced a new reconfigurable block aimed at cryptographic calculations called the cFA to replace LUTs when implementing security primitives. Taking into account only the area of the logic elements, their new cell outperforms traditional FPGAs. The motivation for this master's thesis was to include routing and other factors to carry out a complete comparison.

We want to thank Prof. Dr. Ir Nele Mentens and Dr. Francesco Regazzoni for their help with this master's thesis. They have pointed us in the right direction and supervised us. They also helped us with our questions and ideas.

Contents

List of Tables

List of Figures

Glossary

CLB	Configurable Logic Block. Block of configurable logic inside the FPGA which is connected to the FPGA's routing.
eFPGA	Embedded Field Programmable Gate Array. A small FPGA inside a fabric like a SoC, typically aimed for a single task which can change over time.
FLE	Fracturable Logic Element. A logic element which can be configured to function as multiple smaller logic elements.
IV	Initialization vector. A starting value for a cryptographic algorithm to pseudo-randomize its output. This randomization is important for cryptographic algorithms to prevent patterns to be discovered between messages.
LUT	Lookup table. A configurable component inside a FPGA which can implement logic functions by having a lookup table adressing the logic function's truth table.
Nonce	A random number used for encryption to prevent the use of old messages in replay attacks.

Abstract

Cryptography is an important feature in modern devices, especially in the Internet of Things (IoT). When a deployed cryptographic algorithm is not considered secure anymore, tons of IoT devices are vulnerable to attacks that manipulate the device or retrieve secret information.

In 2018, Mentens et al. proposed to use a small embedded FPGA (eFPGA) for symmetric-key cryptography. The eFPGA allows to update the hardware implementation of the cryptographic algorithm when it is not considered secure anymore. This way, the flexibility properties of software are combined with the efficiency properties of hardware.

In this master's thesis, we compared the eFPGA structure proposed by Mentens et al. to a classical Lookup Table (LUT) based FPGA. The added value of the thesis over the initial experiment conducted by Mentens et al., is the extension of the results from only logic cells only to a fully placed and routed eFPGA structure, based on an open-source design flow.

This thesis concludes that the originally proposed eFPGA structure leads to a reduction of the logic cell area compared to LUT-based FPGAs. However, the overhead of the routing in the novel eFPGA architecture leads to an increase in the total area compared to LUT-based FPGAs.

Abstract in Dutch

Cryptografie is belangrijk in hedendaagse apparaten, specifiek de Internet of Things (IoT) apparaten die lang mee moeten gaan en vaak ondermaats beveiligd zijn. Wanneer een cryptografisch algoritme niet langer bruikbaar is, zullen duizenden van deze IoT-apparaten kwetsbaar zijn voor aanvallen die gegevens aanpassen of stelen.

In 2018 hebben Mentens et al. een voorstel gedaan om een kleine embedded FPGA (eFPGA) te gebruiken voor symmetrische-sleutel cryptografie. Deze eFPGA laat toe om de hardware implementatie van een cryptografisch algoritme te veranderen wanneer het oude algoritme niet meer veilig genoeg is. Dit combineert de flexibele eigenschappen van software met de efficiënte eigenschappen van hardware.

In deze masterproef wordt de eFPGA van Mentens et al. vergeleken met een klassieke Lookup Table (LUT) gebaseerde FPGA. De toegevoegde waarde van de masterproef bovenop het originele experiment van Mentens et al. is dat de resultaten niet gebaseerd zijn op enkel de logische cellen maar dat er een volledige FPGA gemaakt wordt waar ook routing in rekening wordt gebracht.

De oppervlakte voor de logica is kleiner indien de aanpak van Mentens et al. wordt gevolgd, dit bevestigt de resultaten van Mentens et al. Wanneer men echter ook de oppervlakte van de routing mee in rekening neemt dan is de totale oppervlakte van de nieuwe eFPGA groter van die van gelijkaardige LUT gebaseerde FPGAs.

1 Introduction

The security of all our communication depends on encryption. It prevents criminals from reading, modifying or misusing data. In most applications, encryption is mandatory but for certain devices, it can require too many resources or cause a too large overhead.

A possible solution to this issue is the use of dedicated hardware. Dedicated hardware is faster and more energy-efficient than its software counterpart but it has the disadvantage that it cannot be modified in the lifespan of the device.

Almost every modern system has dedicated hardware for one or more encryption algorithms. This gives users the advantages mentioned as long as the encryption algorithm is secure. When an algorithm is superseded by another, the hardware accelerator can no longer be used and the supported encryption algorithm will not enjoy acceleration anymore.

Recent research conducted by Mentens et al. introduces a small FPGA to handle encryption with a new type of cell called the cFA [1]. As other FPGAs, cFAs can do encryption with the same advantages as dedicated hardware and when a different encryption algorithm is used, it can be reprogrammed to handle the new algorithm. In addition to classical FPGAs, the cFA exploits constructions typical of cryptography to be more efficient.

It is expected that a FPGA implementation with cFA cells will be larger and slower compared to ASICs, but it is a major advantage to offer the ability to change the algorithm after a device is deployed.

In the original work of Mentens et al., the cFA cell was compared to a LUT based cell only by looking at the synthesis step of the design flow. The cell is smaller and faster, but has less features compared to a LUT.

In this thesis, the cFA cell is used inside a FPGA and compared to a LUT based FPGA after the placement and routing. We benchmarked some established cryptographic algoritms and a selection of algorithms from the second round candidates of the NIST Lightweight Cryptography Standardization Process. Additionally, a non-cryptographic algorithm is benchmarked.

The literature review gives more information on the cFA cell. Chapter 3 describes the architecture of the FPGA that is designed for this paper to compare LUT and cFA based FPGAs. In the chapter ‘Toolchain‘, the tools used for the benchmarking are presented. Chapter 6 reports results and concludes those results. The next chapter proposes a new cell and chapter 8 concludes this thesis.

2 Literature Review

2.1 cFA cell

Standard FPGA design consist of Lookup Tables (LUTs) for implementing logic functions. These cells are flexible in usage as they can generate any logic function, however due to their area usage and speed, they are often replaced in specific designs. Creating custom cells can improve the area, speed, power consumption and heat production at the cost of flexibility [1].

For cryptographic use, Nele Mentens et al. have proposed a new type of cell to replace traditional LUTs called the cFA (configurable Full Adder) [2]. By implementing only 8 logic functions that are common in cryptology, synthesis results showed that it was possible to reduce the area occupation and to increase the performance. Figure 2.1 shows the design of the cell.

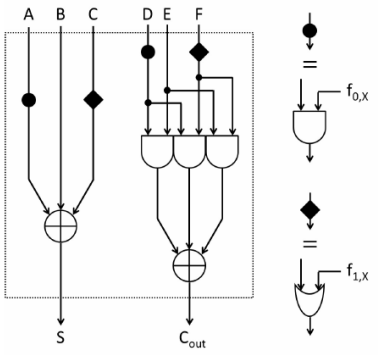

Figure 2.1: Second version of the cFA cell [2]

The cFA cell consists of two separate parts, each configurable in four modes. These modes are operations which are commonly used in crypto algorithms. The cFA reduces the area required to about a tenth and decreases the latency by a third [1] comparing to a LUT which is used inside a classical FPGA.

By design descision, the cFA has similar inputs and outputs as a fracturable LUT with 6 inputs and 2 outputs. These type of LUTs are commonly used in Xilinx FPGAs [3].

2.2 Toolchain

Verilog-To-Routing (VTR) is an open-source toolchain to synthesize and place-and-route Verilog code onto FPGAs. This toolchain consists of the synthesis tool Odin II [4], the place-and-route tool VPR and ABC for optimizations.

Another open-source toolchain is OpenFPGA [5] which is similar to VTR but replaced the Odin II synthesis tool with Yosys [6]. Yosys is a more extensible synthesis tool which includes the UC Berkley's ABC synthesis and verification tool [7][8][9].

Nextpnr is a new open-source place-and-route tool from the same community as Yosys [10] which could replace VPR in the toolchain. At the time of writing, the tool has experimental support for generic FPGA devices and no support for custom cells other than LUTs. While Nextpnr is currently missing these features, it might be a promising tool to use in the future.

Finally, there are commercial tools available to do the synthesis and place-and-route from different companies, these include Intel, Xilinx, Lattice, Synopsys and Cadence. In the toolchain, openly available tools are preferred.

2.3 Comparing baseline and cFA FPGAs

Comparing a FPGA based on cFA cells to a traditional LUT based FPGA must be done in a fair way. Commercial FPGA architectures are kept secret but some details are publicly available.

One of the techniques often used to improve a FPGA's performance is to add a carry chain between logic elements. This can speed up addition and subtraction operations [11]. cFA FPGAs have a dedicated mode to calculate the carry output of an addition, thus it has similar benefits as a carry chain in LUT based FPGAs. Because commercial FPGAs almost always have a carry chain, the carry look ahead mode of the cFA cell should only be used when comparing it to FPGAs with a carry chain.

The standard cell library used by the researchers of the cFA cell is Nangate45 [2]. This library was created for use by universities, educational programs and other research activities. Since 2018, Nangate has been acquired by Silvaco who limits the use of the Nangate libraries [12]. As an alternative, the FreePDK45 standard cell library can be used [13][14]. The Nangate library is based on this standard cell library and it is free to use for research purposes. FreePDK15, the 15nm variant of FreePDK45 is also available, this is a newer and smaller technology but similar commercial technologies are more expensive to fabricate [15]. When comparing the baseline and the cFA FPGA, it is important that the same standard cell library is used as this will have an impact on the area, speed and power consumption.

2.4 Benchmarked algorithms

2.4.1 AES

AES is one of the widely-used encryption algorithms of the past two decades. It was selected after a competition (NIST) as the successor to DES [16]. AES has a block lenght of 128 bits and supports keys of 128, 192 and 256 bits.

2.4.2 Noekeon

Noekeon is a block chipher with a block and key lenght of 128 bits [17]. It is a lightweight algorithm that entered as a candidate in the NESSIE competition in 2000.

2.4.3 Speck

Speck is a block chipher with configurable block and key length [18], an implementation with a block size of 32 bits and a key size of 64 bits is used in this master's thesis.

2.4.4 NIST lightweight cryptography round 2 candidates

The algorithms which will win the National Institute of standards and technology U.S. Deperament of Commerce (NIST) lightweight cryptography competition have a great chance to be used in the industry in the future. This is why some of these algorithms were used to benchmark the LUT based FPGA and the cFA based FPGA. An analysis of the hardware requirements for LUT based FPGAs has already been done [19].

2.4.4.1 ACE

ACE is a lightweight algorithm that has support for both authenticated encryption and hashing. It has 128-bit security [20].

2.4.4.2 Grain-128AEAD

Grain-128AEAD is a 128-bit key stream cipher with a 96-bit nonce IV. The design is based on Grain-128a [21].

2.4.4.3 Subterranean 2.0

Subterranean 2.0 is a stream cipher with a 257-bit state and a single-round permutation. It can be used for hashing, MAC computation, stream encryption and serveral types of authentication [22].

2.4.4.4 WAGE

WAGE is based on the Welch-Gong (WG) stream cipher and uses a 259-bit permutation. It does authenticated encryption with associated data and makes use of a 128-bit key and 128-bit nonce [23].

2.4.4.5 Xoodyak

Xoodyak is a cryptographic algorithm for hashing, encryption, MAC computation and authenticated encryption. It has a claimed security strenght of 128 bits [24].

2.4.5 RISC-V Rocket Chip

The cFA cell is optimized for the most common cryptographic calculations. While the cFA can do the same calculations as the LUT, it will require much more cells to get the same behavior. To get an idea how a non-cryptographic design performs using cFA cells instead of LUTs, the RISC-V Rocket Chip was compiled for the cFA based FPGA. The default configuration Rocket Chip was used, a single core RISC-V processor. The exact configuration is not important as we are comparing the same configuration for both the LUT based FPGA and the cFA based FPGA, although it must be sufficiently large to get a noticable difference in results [25].

2.5 Conclusion

When comparing LUT based FPGAs to cFA based FPGAs, multiple aspects must be considered for achieving a fair comparison. The software must be synthesized and place-and-routed by the same software and with similar settings to prevent optimization advantages for one of the FPGAs. Next, the FPGAs must have a similar layout, ideally only the LUT is replaced by a cFA in the cFA based FPGA. Finally, the standard cell library must be the same for both FPGAs for a fair comparison of area, timing and power usage.

If the cFA outperforms the LUT in cryptographical computations on either area, timing or power usage, the cFA based FPGA can be used as configurable cryptographic hardware inside many processors or microcontrollers. It will allow the cryptographic hardware to be reconfigured when the original encryption algorithm is deprecated or insecure. The processor will have a higher speed and lower power consumption without being tied to a single encryption algorithm.

3 FPGA architecture

3.1 Basic structure

FPGAs are large grids of configurable CLBs, IO pins to interact with signals outside the FPGA and configurable routing to interconnect these CLBs and IO pins.

The CLB is the part of a FPGA which contains the logic. They can be interconnected by reconfigurable routing making it possible to create any logic circuit as long as the FPGA is large enough. CLBs consist of slices which are the smallest unique logic components. The FPGAs created for this thesis use 4 slices inside 1 CLB. Figure 3.1 shows the IO and CLBs of the FPGA layout.

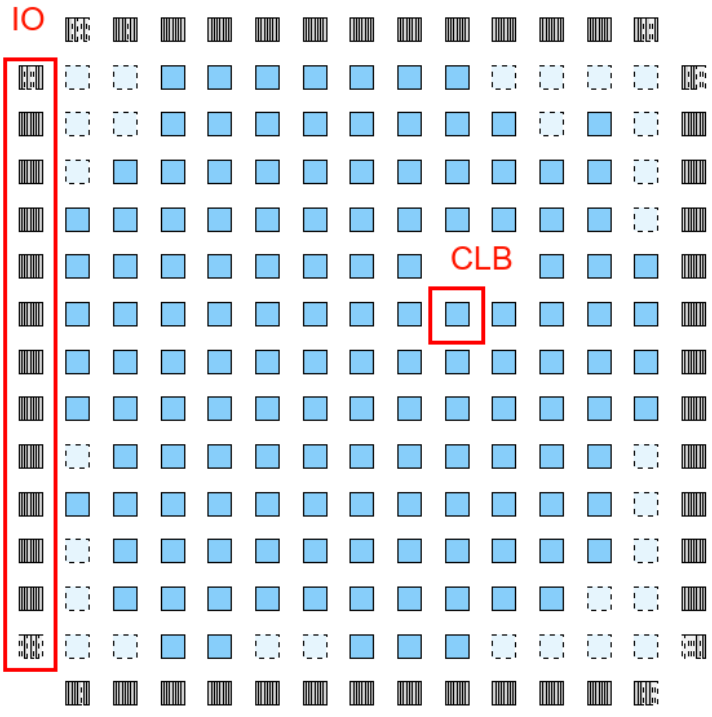

Figure 3.1: FPGA layout

The configurable routing shown in figure 3.2 consists of wire segments running between the CLBs. These wire segments can be connected to the CLB using connection blocks and intersecting wires can be connected in switch blocks.

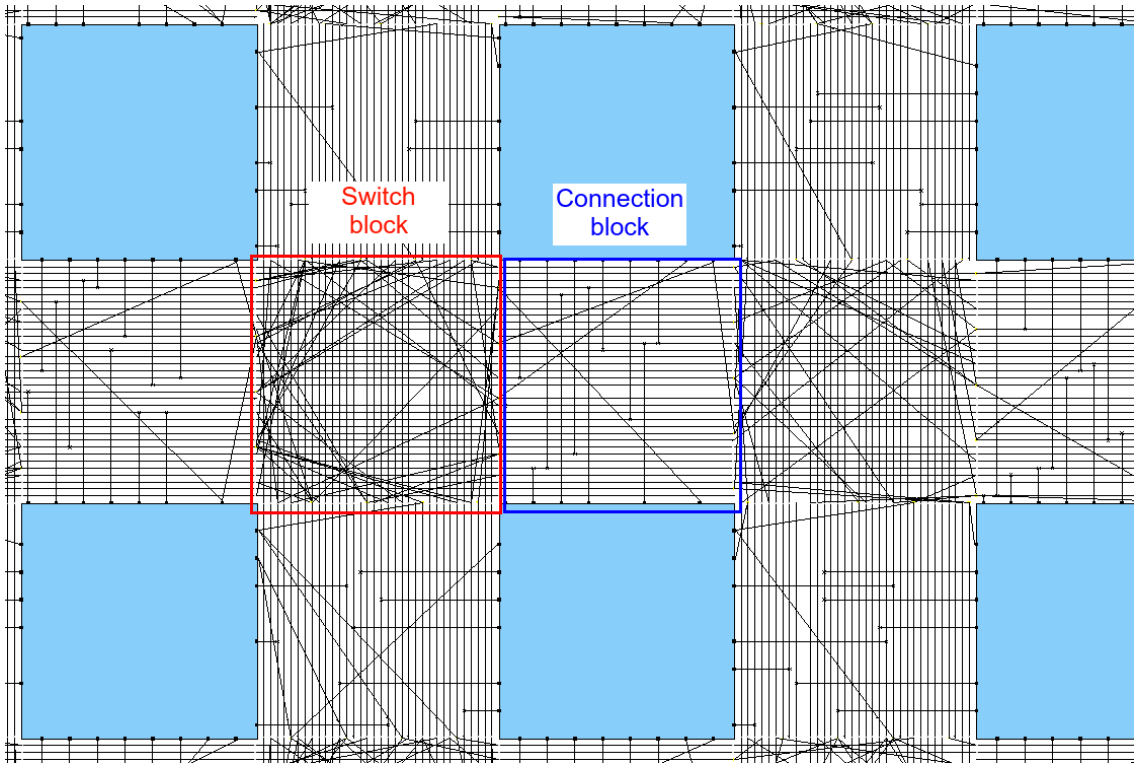

Figure 3.2: FPGA routing layout

3.2 Novel eFPGA slice

3.2.1 Layout

The slice design used is inspired by the slices Xilinx implements in their FPGAs. The inputs of a CLB are connected to a fracturable LUT or a cFA cell with 6 inputs and 2 outputs, they are further described in the next chapters. Both these outputs are connected to a flip-flop which can be bypassed by a multiplexer if the according configuration bit is set.

3.2.2 LUT

The LUT is a 6-input, 2-output fracturable LUT. This enables the LUT to be used as a 6-input 1-output LUT or as two 5-input 2-output LUTs where both 5-input LUTs have the same inputs. The layout of the eFPGA LUT slice is show in figure 3.3.

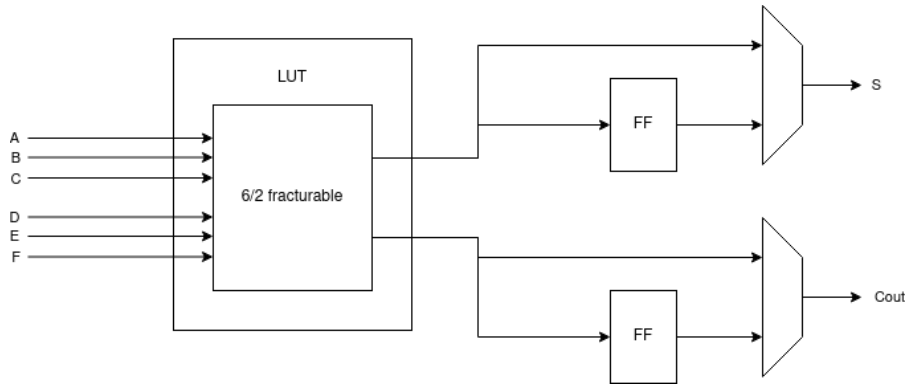

Figure 3.3: LUT slice

3.2.3 cFA cell

The cFA cell has the same number of inputs and outputs as a 6-2 fracturable LUT which is 6 inputs and 2 outputs. The cell has 4 configurable modes in both parts of the cell. The layout of a cFA slice can be seen in figure 3.4.

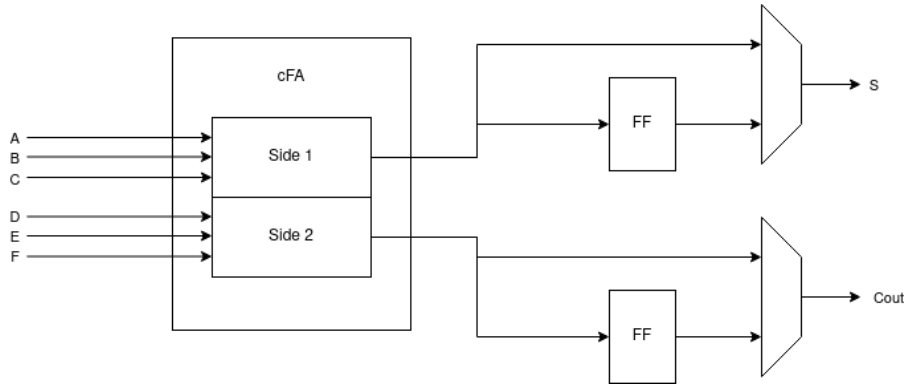

Figure 3.4: cFA slice

3.3 Switch blocks

In both FPGA architectures, identical switch blocks are used. The only difference is in the channel width. Optimal channel width can be different for each algorithm and even vary a little when compiling exactly the same Verilog code with a different random seed. The switch blocks are based on the Wilton architecture because this yields better results for all algorithms compared to the universal and the disjoint switch blocks. The flexibility F_s, which is the amount of outputs an input wire of the switch block can connect to, is set to three. This to not overcomplicate routing inside the switch block.

3.4 Connection blocks

The connection blocks used enable each input of a CLB to be connected to every routing channel. A multiplexer connects one routing channel to each input of the CLB which can be selected with configuration bits [26].

4 Toolchain

4.1 Tools

4.1.1 Yosys

Yosys [7] is an open-source synthesis tool and can be downloaded and modified free of charge. Yosys has mature support for Verilog HDL and can be used for complex real-world designs [8]. It also supports reconfigurable architectures [9]. Because Yosys fulfills all the requirements for our work, it is used as synthesis tool to do the benchmarks of the FPGAs.

4.1.2 GHDL

GHDL is an open-source analyzer, compiler and simulator with experimental support for synthesis [27]. Using the GHDL as Yosys plugin, VHDL support is enabled for Yosys [28]. When the input HDL files are not Verilog but VHDL, Yosys can read these files and export them as Verilog sources.

4.1.3 Verilog-To-Routing

Verilog-To-Routing [26][29] is an open-source place-and-route tool for FPGAs. It has the ability to define custom FPGA architectures and replace LUTs with other types of configurable logic. The tool used is called VPR (Versatile Place and Route), this handles the placement and routing of the FPGA.

4.1.4 Synopsys

The Synopsys tools are used to create an efficient implementation of a CLB and its components. VPR can calculate the total area and timings of an FPGA design by using the area and timings of a CLB provided by Synopsys. The technology library used to create these implementations is NanGate45 [30].

Synopsys is the only tool used that is not open-source. The area and timings of the CLB have to be calculated only once and these values are stored in the configuration files of VPR. Users will not have to use Synopsys to implement their HDL design on the LUT based FPGA and the cFA based FPGA present in the toolchain.

4.2 Toolchain flow

Figure 4.1 shows the toolchain flow and the used files. The user's HDL design that is being compiled for the FPGA needs to be in the Verilog and/or VHDL languages. First, if VHDL code is present, it gets converted to Verilog with the GHDL as Yosys plugin. This converted VHDL code is merged with the user's Verilog code if present. This Verilog code, a modified version of the Nangate library and a configuration file is passed to Yosys, the synthesis tool. In Yosys' configuration file, there is the option to use LUTs or to use the cells from the provided cell library file, this way the design can be synthesized for either the LUT based FPGA or for the cFA based FPGA. Yosys does the synthesis and outputs an Extended Berkley Logic Interchange Format (eblif) file which is passed to VPR, the place-and-route tool, together with a XML that describes the FPGA architecture. There are two XML files included in the toolchain: One for the LUT based FPGA and another for the cFA based FPGA. VPR does the place-and-route and outputs a log file which contains the properties of the FPGA with this algorithm. The output is filtered to get the logic area, routing area, total area, critical path delay, channel width and the FPGA's dimensions. The bitstream size can be calculated by using the channel width, the FPGA's dimensions and the XML with the VPR configuration. This calculation is described in the next chapter.

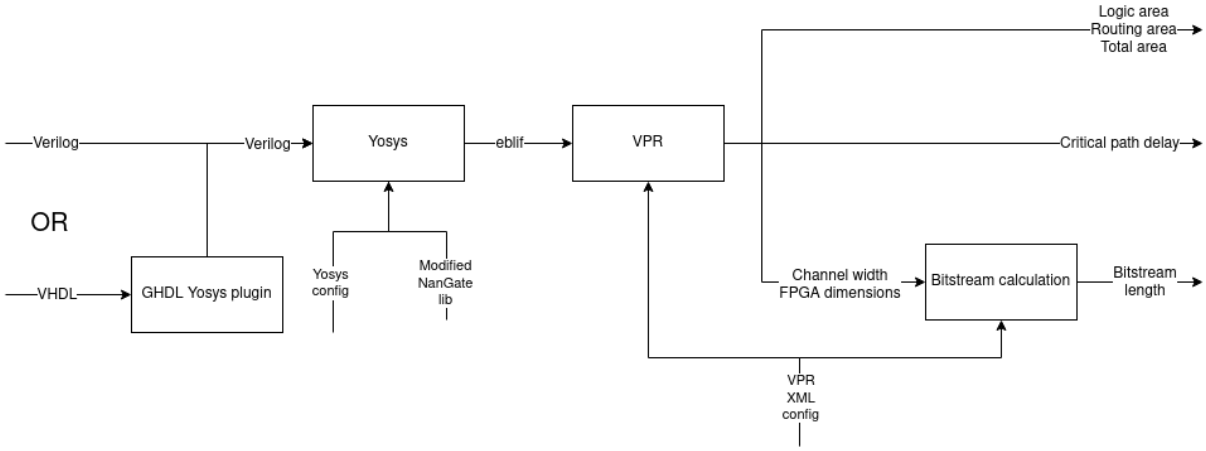

Figure 4.1: Toolchain flow

5 Evaluation methodology

5.1 Measuring area and timing

To calculate the area and timing of a design synthesized for a FPGA, it is required to know the area of a CLB and the timing of the smallest components used by the placement and routing tools.

CLBs consist of Fracturable Logic Elements (FLEs) and those FLEs consist of multiple LUTs in classic FPGAs. The output of those LUTs can optionally be sent through a flip-flop.

Since FPGAs consist of large grids of these CLBs, design tools can use this information to calculate the total area and the timings depending on how many CLBs are used and how they are configured in the design.

Estimating the area usage of a CLB can be done by synthesizing a CLB and then adding up the area usage of every component used inside the CLB. To get the timing information, it is necessary to do synthesis of the smallest components used by the placement and routing tool and then identifying the critical path, a calculation which can be done manually using the synthesis results. While the manual method could provide reasonable results, routing channels are not taken into account.

To calculate the area and timing of the routing channels, the manual method can no longer be used. Instead, a full HDL design of the CLB must be created from which the area and timings can be extracted. An HDL design of the CLB can be synthesized, placed and routed by either a closed source tool like Synopsys or one of the few open source tools available [31] [32].

5.2 Determining the bitstream size

The bitstream of a FPGA is a long binary sequence which contains the configuration of the FPGA. It is typically stored in volatile SRAM inside the FPGA and therefore it must be loaded into the FPGA when power is first applied. This SRAM is connected to the various configurable components in the FPGA like LUTs, cFAs, switch blocks and connection blocks to set their configuration.

The bitstream length is an important criteria when comparing the cFA based FPGA to commercial FPGAs. This is because the configuration bits require SRAM which determine the area usage (and thus the cost) of the FPGA.

To calculate the bitstream length, no external tools are used since they do not support replacing LUTs with different logic. Using certain parameters from the output of the placement and routing

tool VPR, the amount of configuration bits can be calculated. The exact calculation for each component of the FPGA is described in the next chapters. By summing up the bitstream length from all components, the total bitstream length can be obtained.

5.2.1 CLB

The amount of configuration bits is different for LUT based CLBs and cFA based CLBs. When the cFA cell was designed, the amount of configuration bits were taken into account resulting in a much lower usage.

A single 6-input 2-output fracturable LUT has 64 configuration bits for the lookup table itself and one to indicate if it is fractured or not. For each of the two outputs there is an additional configuration bit for the flipflop bypass multiplexer. In total this gives 67 configuration bits per LUT slice and four of these slices add up to a total of 268 configuration bits for a single CLB.

The cFA requires four configuration bits, two for each part. It also has the same multiplexer to bypass the flipflop as used in LUT slices giving 6 bits per cFA slice. Four of these slices add up to 24 bits per CLB.

When comparing the configuration bits of LUT CLBs and cFA CLBs, the cFA CLB has more than ten times less configuration bits.

5.2.2 Switch blocks

In this section, the calculation of the configuration bits required for each switch block is elaborated. When changes are made to the routing architecture, the formula that determines the number of configuration bits for each switch block changes. The formula requires three input variables from the output of VPR (channel width, FPGA width and FPGA height) and one variable from the XML configuration file of VPR (segment length).

The FPGAs used for comparing the LUT based FPGA to the cFA based FPGA all have unidirectional routing which means each wire segment is driven by a single signal at the start of the segment and the wire segment can be connected to multiple inputs at the other intersections. Wire segments are driven by multiplexers whose inputs are other wire segments inside switch blocks or CLB outputs in connection blocks.

There are two types of switch blocks: Full switch blocks in the center of the FPGA which have routing channels coming from all four directions, and partial switch blocks at the sides of the FPGA which have only three incoming routing channels. For full switch blocks, the amount of wires is $channelwidth * 4$, half of these wires are input wires and the other half are output wires. There is a $1/segmentlength$ chance that an output wire starts in the switch block instead of passing through. When this wire starts in the switch block a multiplexer is required to drive the segment, the amount of multiplexers in a switch block can thus be calculated by $outputwires/segmentlength$.

The switch blocks use a modified Wilton pattern when using unidirectional segments. Each segment that ends in the switch block is connected following the Wilton pattern. Segments that pass through are connected following the round-robin scheme. For the bitstream calculation, essentially every input wire connects to three multiplexers of output wires, the amount of wires per multiplexer is calculated by $inputwires*3/multiplexers$. The outputs from the CLBs are also

connected to these multiplexers so two wires should be added to every multiplexer. Finally, the amount of bits per multiplexer are calculated by $\lceil\sqrt{wires/multiplexer}\rceil$ and using the amount of multiplexers per switch block, the FPGA width and the FPGA height, the total bitstream length for all switch blocks is obtained. The formula for both the full switch blocks and the partial switch blocks are shown respectively below.

$$(\frac{channelwidth*4*\frac{1}{2}}{segmentlength})*\left\lceil\sqrt{\frac{channelwidth*4*\frac{1}{2}*3}{\frac{channelwidth*4*\frac{1}{2}}{segmentlength}}+2}\right\rceil*((FPGAwidth-1)*(FPGAheight-1))$$

$$(\frac{channelwidth*3*\frac{1}{2}}{segmentlength})*\left\lceil\sqrt{\frac{channelwidth*\frac{7}{3}*\frac{1}{2}*3}{\frac{channelwidth*3*\frac{1}{2}}{segmentlength}}+2}\right\rceil*(2(FPGAwidth-1)+2(FPGAheight-1))$$

5.2.3 Connection blocks

The connection blocks in the FPGA are used to connect the inputs of the CLB each routing track. The outputs of the CLB are wired to the muxes at the start of each routing wire segment inside the switch blocks, as discussed in the previous chapter. Figure 5.1 shows a diagram of the connection blocks.

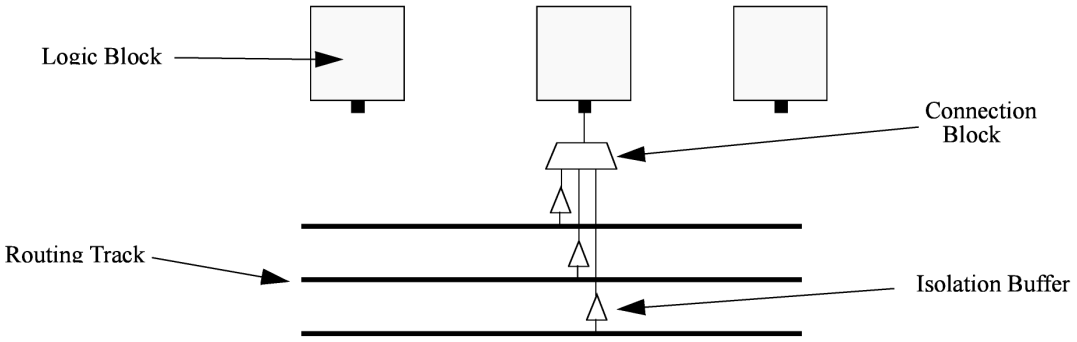

Figure 5.1: Connection block [33]

The amount of selection bits per input pin's multiplexer is the square root of the channel width rounded up, each CLB has 24 input pins and the amount of CLBs is known by the width and height of the FPGA. The final formula for the connection block is shown below.

$$bits/connectionblock = \lceil\sqrt{channelwidth}\rceil * 24 * FPGAwidth * FPGAheight$$

5.2.4 IO pins

The configuration bits of the IO pins are not taken into account in the evaluation. VPR usually generates more IO pins for the cFA based FPGAs, this is because VPR adds one IO pin for each switch block at the side. When the cFA is implemented in a real FPGA design, there will be an equal amount of IO pins as in LUT based FPGAs.

6 Results

6.1 Area

Area is the most important comparison factor since the goal is to replace commercial FPGA structures by dedicated cryptographic eFPGAs in resource-constrained environments. The area usage is one of the most important factors because it influences the cost of the chip.

Table 6.1 shows the area required for each algorithm on the baseline LUT FPGA. Table 6.2 shows the area required by the cFA FPGA. Figure 6.1 shows the relative area used by the logic and routing for the cFA and baseline FPGA compared to the total area of the baseline FPGA.

Table 6.1: Area results of the baseline FPGA

Algorithm	Logic area	Routing area	Total area
Ace	1.32702e+5	1.57487e+6	1.707572e+6
AES	1.45976e+6	1.71404e+7	1.860016e+7
Grain	4.04456e+5	5.46545e+6	5.869910e+6
Noekeon	8.88916e+4	1.20254e+6	1.291432e+6
Speck	3.61916e+4	3.44358e+5	3.805496e+5
Subterranean	1.84132e+5	1.50191e+6	1.686042e+6
Wage	8.50819e+4	7.72584e+5	8.576659e+5
Xoodyak	1.06225e+6	1.32825e+7	1.434475e+7
Rocketchip	2.08905e+6	2.55099e+7	2.759895e+7

Table 6.2: Area results of the cFA based FPGA

Algorithm	Logic area	Routing area	Total area	Percentage cFA/baseline (%)
Ace	6.77100e+4	3.67097e+6	3.73868e+6	218.9%
AES	7.05922e+5	4.45659e+7	4.52718e+7	243.3%
Grain	1.51158e+5	1.13425e+7	1.14937e+7	195.8%
Noekeon	2.28750e+4	1.65812e+6	1.68099e+6	130.1%
Speck	1.22610e+4	6.65534e+5	6.77795e+5	178.1%
Subterranean	3.98025e+4	2.45269e+6	2.49249e+6	147.8%
Wage	5.05995e+4	2.61057e+6	2.66117e+6	310.2%
Xoodyak	4.61068e+5	3.40206e+7	3.44817e+7	240.3%
Rocketchip	1.11264e+6	7.18464e+7	7.29590e+7	264.3%

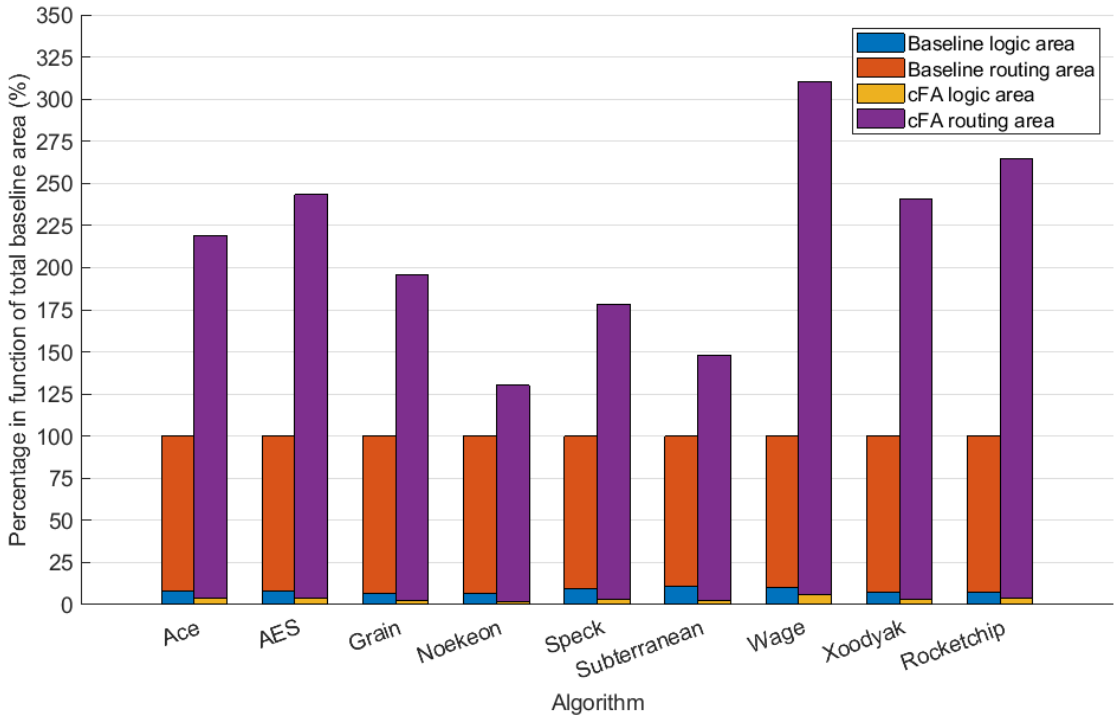

Figure 6.1: Relative routing and logic area usage for baseline and cFA cell

The logic area of the cFA based FPGA is reduced by approximately a factor of 2 compared to the LUT FPGA design. The cFA FPGA has significantly more routing area compared to the LUT FPGA. The total area requirement of the cFA is 130% to 310% higher than the baseline. Notable is the rocketchip which is used as reference for a non-cryptographic application. The expectation was that the cFA would perform significantly worse because the cFA is optimized for cryptographic algorithms and lacks optimizations for general-purpose logic. The only algorithm that performs worse than the Rocketchip is the Wage algorithm.

6.2 Delay

The highest combinatorial delay between two flip-flops in a hardware design determines the frequency of the final circuit. Certain applications demand a certain throughput of data to encrypt. If the maxium frequency of the algorithm on the FPGA does not meet the required throughput, multiple parallel circuits can be added which increase the area.

Table 6.3 shows the maximum frequency of the FPGAs. The last column shows how much of the LUT frequency remains when switching to the cFA based FPGA. Figure 6.2 shows the relative frequency of each algorithm.

Table 6.3: Frequency results of the cFA and LUT based FPGAs

Algorithm	Frequency LUT based FPGA (MHz)	Frequency cFA based FPGA (MHz)	Percentage cFA/baseline (%)
Ace	163.212	106.902	65.4%
AES	61.523	28.104	45.6%
Grain	69.794	62.380	89.4%
Noekeon	134.640	83.442	61.9%
Speck	93.128	65.688	70.5%
Subterranean	208.198	114.603	55.0%
Wage	252.498	71.823	28.4%
Xoodyak	82.876	43.319	52.2%
Rocketchip	48.294	15.630	32.3%

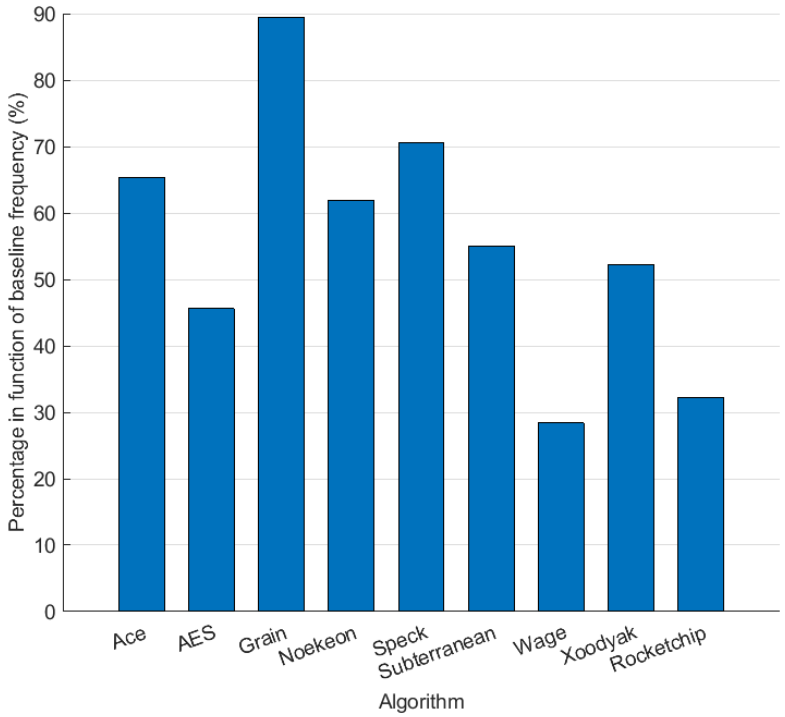

Figure 6.2: Relative routing and logic area usage for baseline and cFA cell

6.3 Configuration bits

The amount of configuration bits is important when designing a FPGA. The configuration bitstream is stored in an SRAM inside the FPGA whichs takes up space. The configuration bitstream must also be loaded into the SRAM when the FPGA is powered up or reprogrammed and this can be done quicker with smaller bitstreams.

Comparing the bitstream of a single LUT to a single cFA, the LUT has 65 configuration bits while the cFA only has four. There are also bits required to program the switch matrices and connection blocks and area results show the cFA has more routing area than the LUT.

Out of the eight tested algorithms, Noekeon and Subterranean have less configuration bits in the cFA based FPGA as shown in table 6.4. On average, the cFA based FPGA uses 142.5% more configuration bits. While the cFA CLBs use significantly less configuration bits, the additional routing required for the cFA results in most algorithms using more. Figure 6.3 shows the relative frequency compared to the baseline FPGA.

Table 6.4: Configuration bit usage

Algorithm	LUT configuration bits	cFA configuration bits	Percentage cFA/baseline (%)
Ace	2.0391e+5	3.1901e+5	156%
AES	1.8792e+6	3.3420e+6	178%
Grain	6.0114e+5	8.5570e+5	142%
Noekeon	1.5897e+5	1.4533e+5	91%
Speck	6.0388e+4	6.7632e+4	112%
Subterranean	2.3749e+5	2.1226e+5	89%
Wage	1.1982e+5	2.3574e+5	197%
Xoodyak	1.4136e+6	2.4743e+6	175%
Rocket chip	2.7258e+6	5.3030e+6	195%

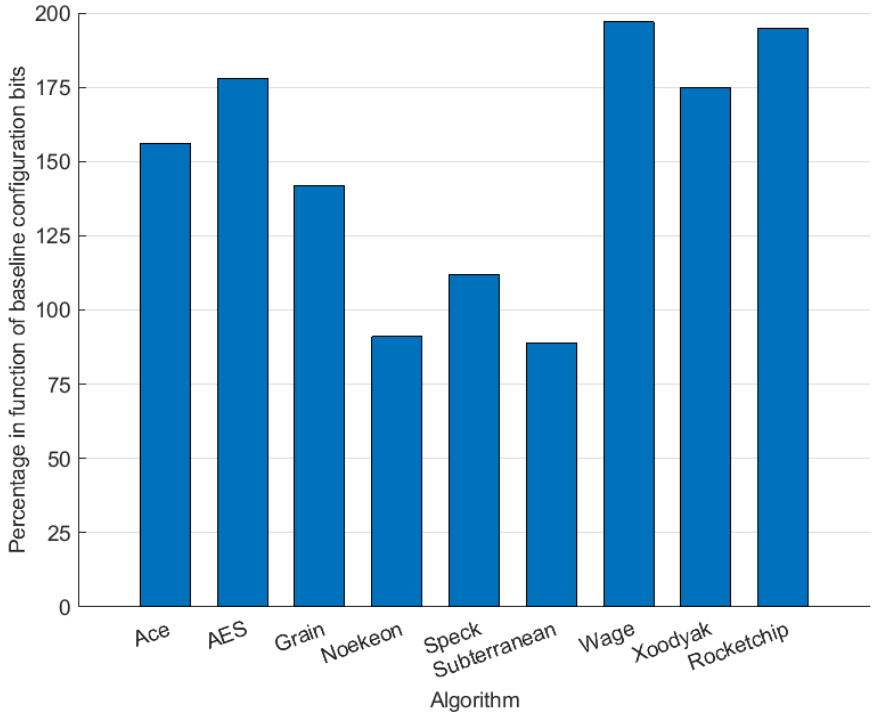

Figure 6.3: Relative routing and logic area usage for baseline and cFA cell

6.4 Conclusion cFA cell

While the comparison results based on logic only from Mentens et al.[2] were favorable for the cFA based cell, the results in this thesis, taking into account routing, show the opposite. One

possible explaination could be that the cFA has 2 parts and Yosys, the synthesis tool, does not use each part of the cFA equally leaving much cell area and its corresponding routing unused. Table 6.5 shows that depending on the algorithm, there is a smaller or large gap in the usage of both parts. Test results show the cryptographic algorithm with the largest imbalance is AES with only 8% of the gates used are on side 1 of the cFA.

Table 6.5: Side of cFA used

Algorithm	Number of used logic gates	Usage side 1 (%)	Usage side 2 (%)
Ace	3424	13.90%	86.10%
AES	33215	7.94%	92.06%
Grain	9102	31.27%	68.73%
Noekeon	1528	63.09%	36.91%
Speck	634	16.09%	83.91%
Subterranean	2657	35.11%	64.89%
Wage	2522	13.60%	86.40%
Xoodyak	25635	21.77%	78.23%

The imbalance of logic functions partially explain the results. To evaluate the impact of the imbalance, the two parts of the cFA were merged by sharing their inputs and choosing one of the two outputs with a multiplexer. The design is show in figure 6.4.

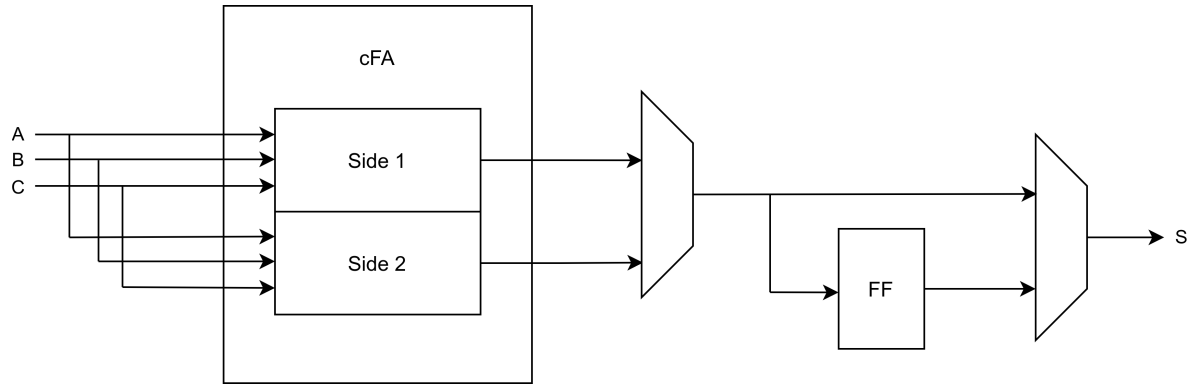

Figure 6.4: cFA slice with 3 inputs and 1 output

In table 6.6, the used routing area of this 3-input 1-output cFA slice is compared with the original 6-input 2-output version. The last column lists the ratio between the routing area of the original cFA and the routing area of the cFA with 3 inputs and 1 output.

Table 6.6: Routing usage combined cFA cell

Algorithm	Routing area usage cFA	Routing area usage 3 in 1 out cFA	Routing area difference (%)
Ace	3.67097e+6	2.95844e+6	80.59%
AES	4.45659e+7	3.28332e+7	73.67%
Grain	1.13425e+7	9.48802e+6	83.65%
Noekeon	1.65812e+6	1.56822e+6	94.58%
Speck	6.65534e+5	5.22763e+5	78.55%
Subterranean	2.45269e+6	2.16206e+6	88.15%
Wage	2.61057e+6	2.09694e+6	80.32%
Xoodyak	3.40206e+7	2.97984e+7	87.59%

The results confirm the impact of the unused parts of the cFA. The algorithm that has the largest imbalance benifits the most from merging of the parts. A solution to the imbalance problem could be to replace cells from the side that is used the most with multiple cells from the other side which together implement the same logic function, this could be done after synthesis. Ideally the synthesis tool would have an option to synthesize with the target to have equal usage of both sides. The current version of Yosys does not have support for this feature. Synthesis tools try to optimize for the best area usage and timing. For configurable cells, the area is always constant. By virtually increasing the area for certain logic functions, the synthesis tool is less likely to pick that function and tries to implement it using other functions. The area difference between the first and second part of the cFA could be increased until the usage is more or less equal.

7 New configurable cell

7.1 Idea

One solution to improve the performance of the cFA cell is to design a new configurable cell. All input should have the ability to do most functionalities. Apendix A shows the Nangate gate usage of the tested algorithms. This is done by running the synthesis of Yosys with the Nangate library modified to have the same area for all logic gates. Based on the analysis in Appendix A, a new configurable cell structure is chosen. This chapter concentrates on this new cell.

7.2 Required gates

Simple logic gates like AND and XOR are mostly used with 2 inputs. On average, the 2-input 1-output multiplexer is the second most used logic element.

7.3 Proposed cell

Figure 7.1 shows the diagram of the new cell. The most important feature is that both sides are very similair. They both have AND, OR and EXOR, with an optional inversion.

Only one side has a multiplexer and the select pin is the B1 pin from the other side. When this is used, the B1 pin can still be combined with the A1 for the normal gates or the A1 pin can be buffered or inverted. The buffer function is also required to connect to the input of the flip-flop directly from the output of another flip-flop if multiple flip-flops should be put in series.

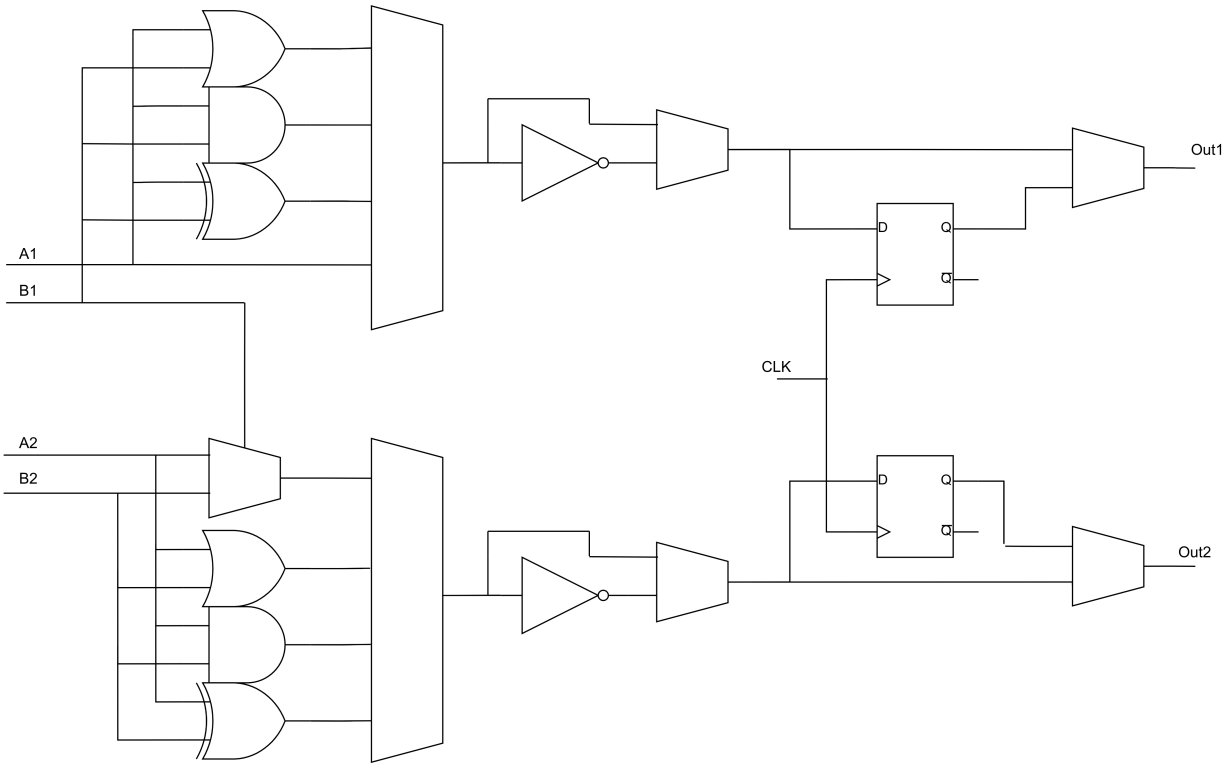

Figure 7.1: The newly proposed slice

7.4 Results

7.4.1 Area

Table 7.1 shows the total area usage of the cell. Only with the Noekeon algorithm there is a very slight advantage towards the proposed cell. All other algorithms require a maximum 180% of the baseline area.

Figure 7.2 compares the baseline, cFA and the propossed cell inside a FPGA. The proposed cell achieves better results than the cFA cell, especially in routing. It is still worse in almost all cases than the baseline FPGA. While the original research by Mentens et al. proposes a fine grained FPGA architecture, it could be worth to explore larger and more specialized cells or blocks.

Table 7.1: Area usage proposed cell

Algorithm	Logic area	Routing area	Total area	Percentage new cell/Baseline (%)
Ace	6.52236e+4	2.57341e+6	2.63863e+6	154.50%
AES	5.97667e+5	2.75432e+7	2.81409e+7	151.20%
Grain	1.90349e+5	9.02002e+6	9.21037e+6	156.90%
Noekeon	2.89885e+4	1.21111e+6	1.24010e+6	96.00%
Speck	1.22757e+4	4.87184e+5	4.99460e+5	131.20%
Subterranean	5.01380e+4	2.12473e+6	2.17487e+6	128.90%
Wage	3.88977e+4	1.47667e+6	1.51557e+6	176.70%
Xoodyak	4.41050e+5	2.24784e+7	2.29195e+7	159.70%
Rocketchip	8.87946e+5	3.83142e+7	3.92021e+7	142.00%

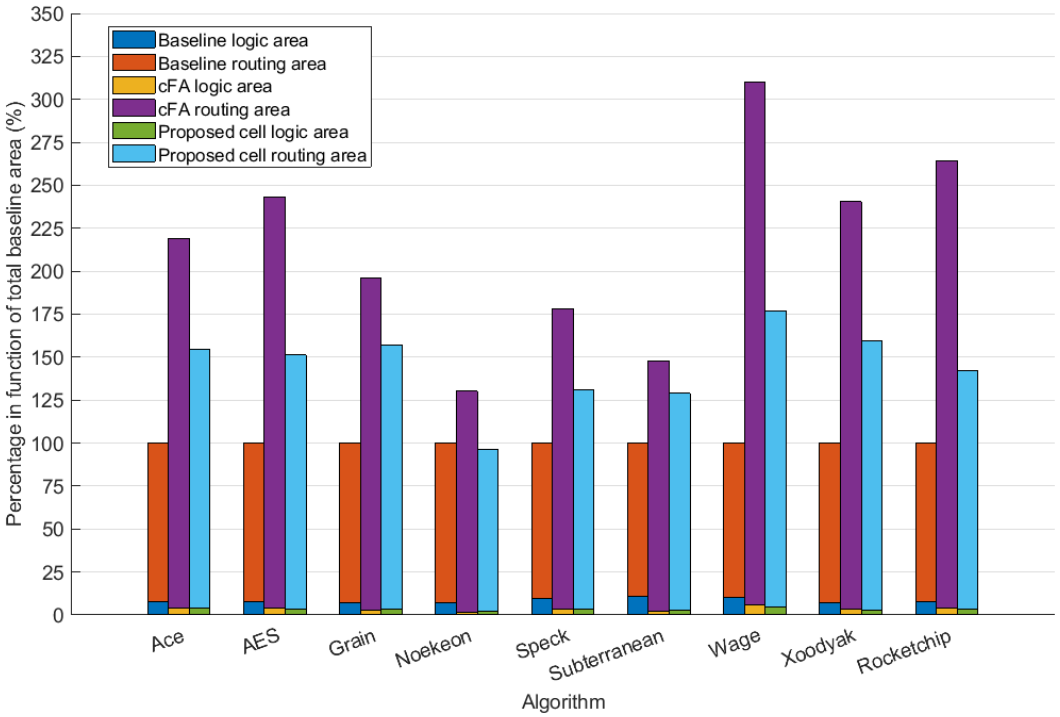

Figure 7.2: Relative routing and logic area usage for baseline and cFA cell

7.4.2 Frequency

Table 7.2 shows the frequency of the propsed cell compared to the baseline. The frequency is between 30% and 74% of the baseline LUT based FPGA for cryptographic algorithms. With the non-cryptographic Rocketchip only 18% of the frequency remains. Figure 7.3 shows that the frequency of the proposed cell is similair to the cFA cell for some algorithms, but significantly lower for Speck and the Rocketchip.

Table 7.2: Frequency proposed cell

Algorithm	Frequency using the new cell (MHz)	Percentage new cell/baseline(%)
Ace	98.165	60.1%
AES	29.968	48.7%
Grain	51.522	73.8%
Noekeon	82.485	61.2%
Speck	36.314	38.9%
Subterranean	121.845	58.5%
Wage	77.225	30.5%
Xoodyak	43.216	52.1%
Rocketchip	8.635	17.8%

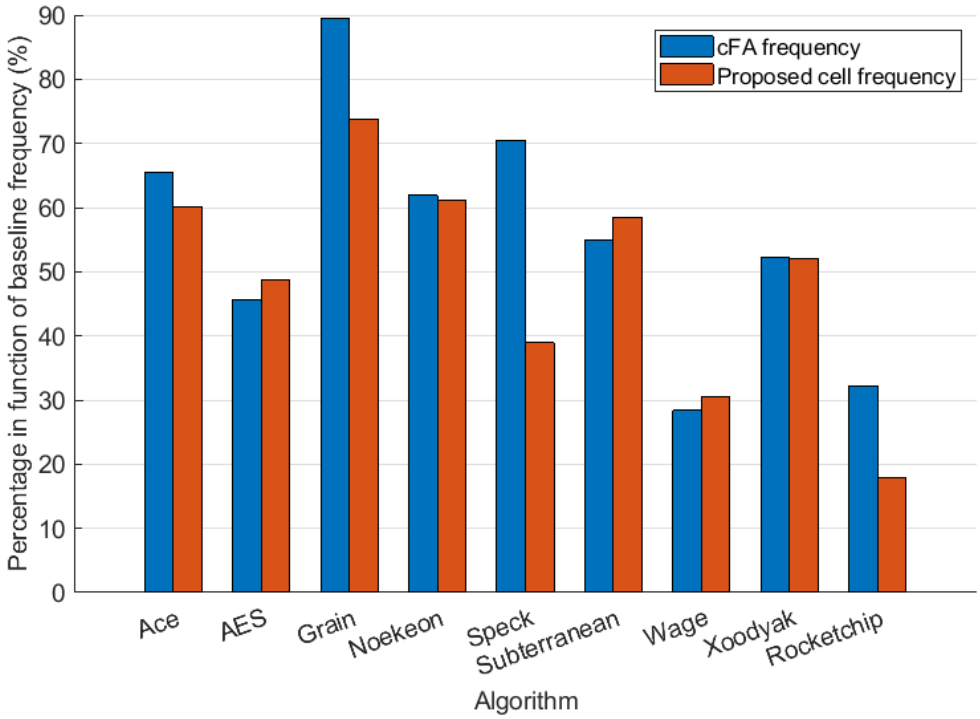

Figure 7.3: Relative frequency of cFA and proposed cell

7.4.3 Bitstream

Table 7.3 shows that the propsed cell requires between 73% and 122% of the baseline bitstream length. Figure 7.4 shows that the proposed cell requires less bits then the cFA cell.

Table 7.3: Configuration bit usage

Algorithm	LUT configuration bits	cFA configuration bits	Percentage cFA/baseline (%)
Ace	2.0390e+5	2.3650e+5	116%
AES	1.8792e+6	2.1944e+6	117%
Grain	6.0114e+5	7.3468e+5	122%
Noekeon	1.5897e+5	1.1585e+5	73%
Speck	6.0388e+4	5.2116e+4	86%
Subterranean	2.3749e+5	1.9739e+5	83%
Wage	1.1982e+5	1.4407e+5	120%
Xoodyak	1.4136e+6	1.7075e+6	120%
Rocket chip	2.7259e+6	2.9898e+6	110%

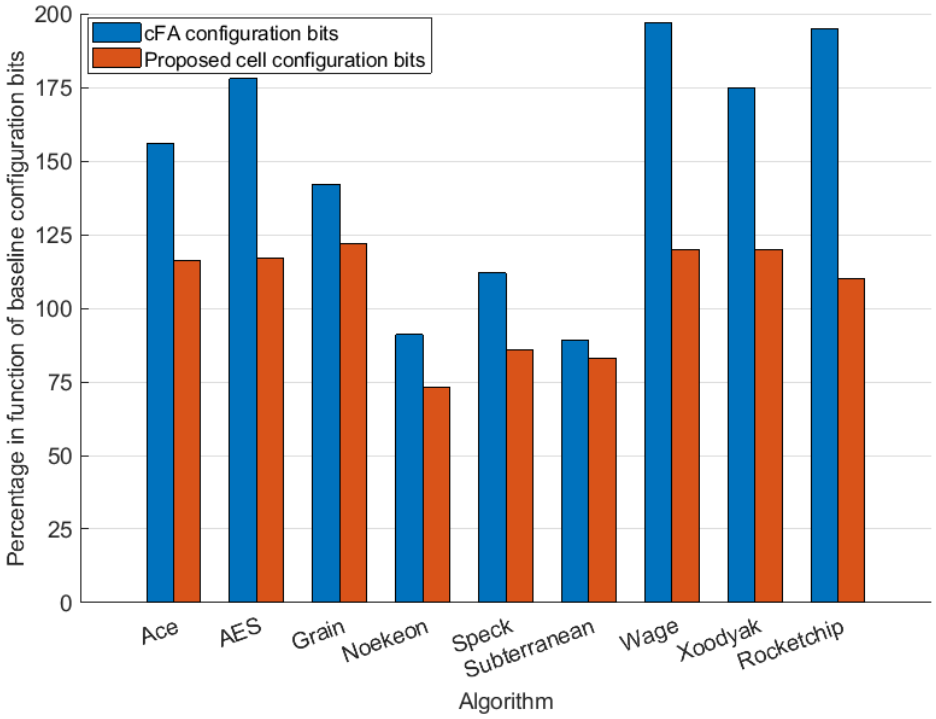

Figure 7.4: Relative frequency of cFA and proposed cell

7.5 Conclusion

By designing a new cell knowing the problems of the cFA cell with the used tools, a better results for the area and bitstream length is achieved. The frequency was worse than the cFA cell in certain algorithms.

8 Conclusion

This master's thesis analyzed the eFPGA structure proposed by Mentens et al. for configurable cryptographic implementations. Testing concluded that the cFA cell does not perform as expected compared to the traditionally used LUT when routing is taken into account in addition to logic cells. This is partially due to the way in which the structure is handled by design tools. The cFA consist of 2 separate parts and because one part has a higher usage than the other, the cFA cell is not used efficiently. If the tools had the ability to effeciently use both parts, the area of the cFA based FPGA would be reduced.

The logic area is on average two times higher when using a FPGA based on the cFA cell compared to a FPGA based on LUTs. Due to the increased routing area usage, the total area of a cFA based FPGA is 130% to 310% larger than a LUT based FPGA.

In an attempt to improve the cFA's performance, its connections to the FPGA's routing were modified. By sharing the inputs' and outputs' routing, a reduction of 5% to 20% was achieved.

The amount of configuration bits influences the area of a FPGA since more configuration bits require more SRAM memory. Depending on the algorithm, the cFA based FPGA can use less configuration bits than the LUT based FPGA. The configuration bit usage of the cFA based FPGA compared to the LUT based FPGA is between 89% and 197%.

The timing results show the cFA based FPGA have 28% to 90% of the LUT based FPGA's frequency. This is due to the delay from the additional routing.

With additional support from the synthesis tool the cFA would still perform worse than the LUT when including routing, this is proven by connecting both sets of inputs together. This reduces the routing area, but it still requires more area than the LUT based FPGA. A breakthrough in routing could make the cFA better than the LUT for cryptographic applications.

The proposed cell achieves better results than the cFA cell in our tests. Better results are achieved because the proposed cell is better supported by the toolchain.

Because the majority of the area on our FPGA is due to the routing, there is need for a better way of routing fain grained FPGA architectures. Otherwise, the area usage of the cell itself is not significant enough to make a big difference.

References

[1] J. Kim and J. Anderson, "Synthesizable standard cell fpga fabrics targetable by the verilog-to-routing cad flow," *ACM transactions on reconfigurable technology and systems*, vol. 10, no. 2, pp. 1–23, 2017.

[2] N. Mentens, E. Charbon, and F. Regazzoni, "Rethinking secure fpgas: Towards a cryptography-friendly configurable cell architecture and its automated design flow," pp. 215–215, IEEE Computer Society, 2018.

[3] Xilinx, *7 Series FPGAs Configurable Logic Block*, 2016.

[4] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin ii - an open-source verilog hdl synthesis tool for cad research," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 149–156, IEEE, 2010.

[5] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon, "Openfpga: An open-source framework for agile prototyping customizable fpgas," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, 2020.

[6] E. Hung, "Mind the (synthesis) gap: Examining where academic fpga tools lag behind industry," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, Imperial College, 2015.

[7] C. Wolf, "Yosys open synthesis suite." http://www.clifford.at/yosys.

[8] J. G. Clifford Wolf, "Yosys - a free verilog synthesis suite," in *Proceedings of Austrochip*, 2013.

[9] J. Glaser and C. Wolf, "Methodology and example-driven interconnectsynthesis for designing heterogeneouscoarse-grain reconfigurable architectures," in *Jan Haase, editor, Models, Methods, and Tools for Complex Chip Design. Lecture Notes in Electrical Engineering*, vol. 265, p. 21, 2013.

[10] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanović, "Yosys+nextpnr: an open source framework from verilog to bitstream for commercial fpgas," 2019.

[11] S. Hauck, M. Hosler, and T. Fry, "High-performance carry chains for fpga's," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 2, pp. 138–147, 2000.

[12] "Silvaco completes acquisition of nangate." https://silvaco.com/news/silvaco-completes-acquisition-of-nangate/, 2018.

[13] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "Freepdk: An open-source variation-aware de-

sign kit," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, pp. 173–174, IEEE, 2007.

[14] J. Stine, J. Chen, I. Castellanos, G. Sundararajan, M. Qayam, P. Kumar, J. Remington, and S. Sohoni, "Freepdk v2.0: Transitioning vlsi education towards nanometer variation-aware designs," in *2009 IEEE International Conference on Microelectronic Systems Education*, pp. 100–103, IEEE, 2009.

[15] M. Martins, J. Matos, R. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on international symposium on physical design*, vol. 29- of *ISPD '15*, pp. 171–178, ACM, 2015.

[16] J. Daemen and V. Rijmen, *The Design of Rijndael: The Advanced Encryption Standard (AES)*. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2020.

[17] D. Joan, P. Michaël, V. A. Gilles, and V. Rijmen, "Nessie proposal: Noekeon," 10 2000.

[18] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers." Cryptology ePrint Archive, Report 2013/404, 2013. `https://eprint.iacr.org/2013/404`.

[19] B. Ovilla-Martinez, C. Mancillas-Lopez, A. F. Martinez-Herrera, and J. A. Bernal-Gutierrez, "Fpga implementation of some second round nist lightweight cryptography candidates," *Electronics (Basel)*, vol. 9, no. 11, p. 1940, 2020.

[20] M. Aagaard, R. AlTawy, G. Gong, K. Mandal, and R. Rohit, "Ace: An authenticated encryption and hash algorithm," *Submission to NIST-LWC*, 2019.

[21] M. Hell, T. Johansson, W. Meier, J. Sönnerup, and H. Yoshida, "Grain-128aead-a lightweight aead stream cipher," *NIST Lightweight Cryptography, Round*, vol. 1, 2019.

[22] J. Daemen, P. M. C. Massolino, A. Mehrdad, and Y. Rotella, "The subterranean 2.0 cipher suite," *IACR Transactions on Symmetric Cryptology*, vol. 2020, pp. 262–294, Jun. 2020.

[23] M. Aagaard, R. AlTawy, G. Gong, K. Mandal, R. Rohit, and N. Zidaric, "Wage: An authenticated cipher," *Submission to NIST Lightweight Cryptography Standardization Project (announced as round 2 candidate on August 30, 2019)*, 2019.

[24] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodyak, a lightweight cryptographic scheme," 2020.

[25] "Rocket chip generator." https://github.com/chipsalliance/rocket-chip.

[26] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. ElDafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "Vtr 8: High performance cad and customizable fpga architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, 2020.

[27] "Ghdl." https://github.com/ghdl/ghdl.

[28] "ghdl-yosys-plugin." https://github.com/ghdl/ghdl-yosys-plugin.

[29] "verilog-to-routing." https://github.com/verilog-to-routing/vtr-verilog-to-routing.

[30] "Nangate cell library." https://silvaco.com/services/library-design/.

[31] K. S. Roy, K. Abhiram, M. A. Sumanth, J. Jaishankar, P. Abhishek, B. N. Prabhat, and L. G. Teja, "Development of graphical user interface for open source vlsi digital synthesis tool qflow," *International journal of engineering & technology (Dubai)*, vol. 7, no. 2, p. 710, 2018.

[32] G. Ahmed, Alaa and S. Mohamed, "Openlane: The open-source digital asicimplementation flow," 2020.

[33] "Vtr architecture reference." https://docs.verilogtorouting.org/en/latest/arch/reference/.

List of Appendices

Apendix A - usage Nangate cells

Algorithm	ace	aes	grain	Noek-eon	speck	subter-ranean	wage	xood-yak	Average Per-cent-age
XNOR2_X1	17.47%	9.03%	45.79%	49.90%	11.38%	38.50%	14.44%	24.15%	26.33%
MUX2_X1	3.31%	16.07%	2.04%	5.47%	19.36%	1.65%	30.74%	4.33%	10.37%
DFFR_X2	0.00%	18.04%	5.49%	0.00%	20.16%	0.05%	22.47%	14.78%	10.12%
XOR2_X1	3.89%	3.50%	2.47%	17.58%	12.18%	4.26%	7.18%	6.34%	7.17%
AND2_X1	2.48%	3.53%	23.35%	2.96%	3.39%	12.83%	2.28%	3.26%	6.76%
INV_X1	0.88%	1.97%	5.87%	1.22%	20.96%	0.10%	1.69%	11.87%	5.57%
DFF_X2	15.91%	0.00%	0.00%	8.76%	0.00%	12.88%	0.00%	0.00%	4.69%
NAND2_X1	5.40%	3.38%	8.74%	7.53%	2.99%	0.85%	4.14%	1.18%	4.28%
OAI21_X1	18.73%	3.99%	0.23%	2.90%	2.79%	1.75%	2.87%	0.07%	4.17%
NOR2_X1	5.01%	0.85%	0.26%	1.61%	1.80%	12.93%	1.69%	2.12%	3.28%
AND3_X1	0.10%	3.53%	2.26%	0.19%	0.60%	0.05%	1.77%	11.52%	2.50%
AOI21_X1	2.48%	2.75%	0.32%	0.26%	2.79%	0.90%	3.21%	5.79%	2.31%
AOI22_X1	4.28%	10.00%	0.35%	0.00%	0.20%	0.00%	0.00%	0.37%	1.90%
AOI211_X1	0.05%	1.01%	0.07%	0.06%	0.20%	11.98%	0.93%	0.02%	1.79%
AOI222_X1	5.99%	3.18%	0.00%	0.00%	0.00%	0.00%	0.00%	1.07%	1.28%
OAI33_X1	0.00%	0.02%	0.00%	0.00%	0.00%	0.00%	0.00%	8.98%	1.12%
AOI221_X1	5.89%	1.74%	0.00%	0.00%	0.00%	0.00%	0.17%	0.01%	0.98%
NAND3_X1	0.19%	3.87%	1.21%	0.00%	0.20%	0.00%	2.20%	0.06%	0.97%
OR2_X1	1.75%	0.89%	0.07%	1.22%	0.00%	0.10%	0.84%	2.29%	0.90%
AND4_X1	0.00%	5.67%	0.63%	0.00%	0.20%	0.00%	0.00%	0.02%	0.82%
NOR3_X1	2.97%	0.24%	0.02%	0.00%	0.00%	1.00%	0.00%	1.65%	0.73%
OAI211_X1	2.77%	1.70%	0.00%	0.13%	0.20%	0.05%	0.59%	0.02%	0.68%
NAND4_X1	0.00%	1.65%	0.46%	0.00%	0.20%	0.00%	0.00%	0.01%	0.29%
OAI22_X1	0.24%	1.53%	0.16%	0.06%	0.00%	0.00%	0.17%	0.02%	0.27%
OR3_X1	0.05%	0.28%	0.14%	0.06%	0.00%	0.05%	1.44%	0.03%	0.26%
BUF_X1	0.00%	0.00%	0.01%	0.00%	0.00%	0.05%	1.18%	0.00%	0.16%
NOR4_X1	0.00%	0.67%	0.00%	0.00%	0.40%	0.00%	0.00%	0.01%	0.14%
OAI221_X1	0.15%	0.64%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.10%
OR4_X1	0.00%	0.24%	0.06%	0.06%	0.00%	0.00%	0.00%	0.02%	0.05%
DFFS_X2	0.00%	0.04%	0.01%	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%
OAI222_X1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%