DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

An FPGA Implementation of Arbiter PUF with 4x4 Switch Blocks

CAN AKNESIL

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

An FPGA Implementation of Arbiter PUF with 4×4 Switch Blocks

Can Aknesil

School of Electrical Engineering and Computer Science
Royal Institute of Technology (KTH), Stockholm, Sweden
aknesil@kth.se

Master of Science Thesis in Embedded Systems
Supervisor: Elena Dubrova

June 15, 2020

Abstract

Theft of services, private information, and intellectual property have become significant dangers to the general public and industry. Cryptographic algorithms are used for protection against these dangers. All cryptographic algorithms rely on secret keys that should be generated by an unpredictable process and securely stored. The keys are usually stored in a memory, e.g. Flash or fuses. Therefore, the strength of cryptographic protection relies upon the ability of an attacker to extract the keys from the hardware. Modern hardware implementation methods are very advanced, weakening cryptographic algorithms against physical attacks. Finally, memories that provide extra security are expensive to be used in Integrated Circuits (ICs).

As a solution to the memory key storage problem, Physically Unclonable Functions (PUFs) have been proposed. A PUF is an electronic circuit that evaluates responses of hardware to given input stimuli. Due to manufacturing process variations, every IC has different characteristics at the analog level. These variations lead to measurable differences, hence different responses of PUFs implemented on different IC chips.

In this thesis, we are implementing recently proposed 4×4 Arbiter Physically Unclonable Function (APUF) on Field-Programmable Gate Arrays (FPGAs), performing statistical analysis including uniformity, reliability, and uniqueness, comparing hardware overhead of our FPGA design to other APUF variants, providing a mathematical model using homogeneous coordinates, and proposing methods that enable usage of our PUF in real-world applications. We selected this particular type of PUF because it is claimed to be more area efficient than its alternatives while providing strong security and reconfigurability.

According to our analysis, the presented 4×4 APUF design is suitable for many security applications, including identification, authentication, encryption, and key generation. Furthermore, its FPGA area is considerably smaller than the area of 2×2 APUF variants accepting challenges of the same size. However, since uniqueness of our design is lower than desirable, to be used in security applications, our PUF requires repeated invocations and generation of larger keys by combining many responses, thus additional computation during runtime.

Sammanfattning

Stöld av tjänster, privat information och immateriell egendom har blivit betydande faror för allmänheten och industrin. Kryptografiska algoritmer används för att skydda mot dessa faror. Alla kryptografiska algoritmer förlitar sig på hemliga nycklar som ska genereras genom en oförutsägbar process och säkert lagras. Tangenterna lagras vanligtvis i ett minne, t.ex. Flash eller säkringar. Därför beror styrkan på kryptografisk säkerhet på en angripares förmåga att extrahera nycklarna från hårdvaran. Moderna fysiska metoder på hårdvara är mycket avancerade och försvagar kryptografiska algoritmer mot fysiska attacker. Slutligen är minnen som ger extra säkerhet dyra att använda i Integrated Circuits (ICs).

Som en lösning på minnesnyckellagringsproblemet har fysiskt oklonbara funktioner (PUF) föreslagits. En PUF är en elektronisk krets som utvärderar svar från hårdvara på givna input stimuli. På grund av variationer i tillverkningsprocessen har varje IC olika egenskaper på den analoga nivån. Dessa variationer leder till mätbara skillnader, därmed olika svar från PUF: er implementerade på olika IC-chips.

I den här avhandlingen implementerar vi nyligen föreslagna 4×4 Arbiter Physically Unclonable Function (APUF) på fältprogrammerbara gate-arrayer (FPGAs), och utför statistisk analys inklusive enhetlighet, tillförlitlighet och unikhet, jämförande hårdvarukostnader för vår FPGA-design med andra APUF varianter, ger en matematisk modell med homogena koordinater och föreslår metoder som möjliggör användning av vår PUF i verkliga applikationer. Vi valde den här typen av PUF eftersom den påstås vara mer areaeffektiv än dess alternativ samtidigt som den ger stark säkerhet och omkonfigurerbarhet.

Enligt vår analys är den presenterade 4×4 APUF-designen lämplig för många säkerhetsapplikationer, inklusive identifiering, autentisering, kryptering och nyckelgenerering. Dessutom är dess FPGA resursanvändning avsevärt mindre än området för 2×2 APUF-varianter som accepterar utmaningar av samma storlek. Eftersom unikhet med vår design är lägre än önskvärt, för att användas i säkerhetsapplikationer, kräver vår PUF upprepade åkallelser och generering av större nycklar genom att kombinera många svar, därmed ytterligare beräkning under körning.

Contents

Glossary

6-LUT	6 Input Loop-Up Table. 32
APUF	Arbiter Physically Unclonable Function. 2, 9–14, 16, 18–21, 23–27, 31–33, 35, 36
ASIC	Application-Specific Integrated Circuit. 9
CW	ChipWhisperer. 11, 20, 24
FPGA	Field-Programmable Gate Array. 2, 9, 11, 14, 15, 20, 21, 24–26, 29, 30, 36
FSM	Finite State Machine. 19
IC	Integrated Circuit. 2, 15, 16
ML	Machine Learning. 11, 13, 14, 36
PUF	Physically Unclonable Function. 2, 9–14, 34, 35
RFID	Radio-Frequency Identification. 10
RNG	Random Number Generator. 9, 10
SSH	Secure Shell. 9
TRNG	True Random Number Generator. 35

List of Figures

List of Tables

1 Introduction

Many security applications rely on secret keys that should be generated by an unpredictable process and stored securely. Secret keys are generally stored in non-volatile memories, disks, or they are hard-wired in the hardware in a way that only allows access to authorized people. One example is private keys used during encrypted communication, such as with the Secure Shell (SSH) protocol. These keys are stored in the disk and their access is controlled by the file permissions. Another example is smart cards (credit cards, SIM cards, etc.) that store a unique key used to identify the owner of the card.

However, attackers can extract the secret key via physical attacks such as micro-probing, laser cutting, glitch attacks, and power analysis. It is also possible to clone the Random Number Generator (RNG) used to generate the secret keys by analyzing previously generated ones and guess the future keys. Even though it is possible to achieve strong randomness via computational complexity, hardware implementations of RNGs can be predicted with reverse engineering [16]. Consequently, there is a need to improve security at the hardware level.

One solution of secure generation and storage of secret keys is PUFs. PUFs exploit random manufacturing process variations of electronic devices such as Application-Specific Integrated Circuits (ASICs) and FPGAs to generate device-specific keys that cannot be cloned [16]. They are built into a chip during manufacturing. This eliminates the need for manual per-device configuration. The keys are generated only when required and do not remain stored on-chip. This provides a higher resistance to physical attacks. Furthermore, they cannot be cloned because it is nearly impossible to fabricate an electronic device with the same manufacturing imperfections.

For many applications, PUFs should be reconfigurable so that they can generate different responses to different inputs, where each input represents a different configuration.

Previous Work A model of a new type of PUF, an APUF with 4×4 switch blocks, is proposed in [8]. It is based on the conventional APUF [16] that uses 2x2 switch blocks. The new design is claimed to provide better resistance against modeling attacks while keeping hardware overhead and computation time low.[1]

Our Contribution The contributions of this thesis can be summarized as follows:

- An FPGA implementation of APUF with 4×4 switch blocks.
- Statistical analysis on our implementation including uniformity, reliability, and uniqueness. We also extend our analysis to newly discovered behavior due to unequal delays leading to the arbiter.

[1] The comparison with other PUF designs are performed for Xilinx series 7 FPGAs [10] which are also used for our implementation.

- Verification of previously anticipated [10] hardware overhead.
- A model for 4×4 APUF based on homogeneous coordinates, which can be generalized to switch blocks of arbitrary size.
- Methods to enable the usage of the presented 4×4 APUF design in real-world applications.

Research question Is APUF with 4×4 switch blocks realizable on FPGAs with a sufficient amount of exploitation of manufacturing differences to be used in real-world applications?

1.1 Applications of PUFs

PUFs can be thought of as a unique fingerprint for electronic devices. There are numerous applications of PUFs [3]:

Identification A PUF is embedded into a device, such as a Radio-Frequency Identification (RFID) tag, intended to store an ID. This can reduce the cost since an internal non-volatile memory will not be needed.

Authentication A user owns a device containing a PUF that is used for authentication via a server. The server sends an input to the user who uses it to generate a response from the PUF and sends the response back to the server. Then, the server compares the received response to the one in its database to check its validity. In this scenario, a different input should be used for every authentication to prevent the attackers, listening to the communication between the user and the server, to impersonate the user. This necessitates the definition of “lifetime” of the PUF, which should expire before disclosure of a sufficient amount of input-response pairs to break the security.

Random Number Generation A PUF can be used as an unpredictable RNG that behaves differently on every device. Generated numbers can be used as secret keys if desired.

1.2 4×4 Arbiter PUF

A general APUF constitutes of symmetrically placed paths. It is important that the propagation times of a signal through all of these paths are as close as possible. An APUF is used by simultaneously sending a signal, called a “stimuli”, to each of these paths and comparing the propagation delays with an arbiter that generates a response representing the order of the delays. In real life, due to the random manufacturing differences, the delays slightly differ resulting in unique responses for every chip.

4×4 APUF proposed in [8] contains a structure called “4×4 switch block” as the building block. A 4×4 switch block is capable of mapping 4 inputs to

4 outputs in every 24 (4!) possible ways, and is controlled by a 5-bit selector called the “challenge”. The most important property of a switch block is that all the 16 (4^2) paths connecting one input to one output (among which 4 of them are selected according to the challenge) has theoretically the same delays, resulting in 16 distinct delay values per chip when implemented in real life.

A 4×4 APUF is constructed by concatenating multiple switch blocks, creating 4 long paths whose delays are reconfigured via a 5-bit challenge per switch block. A high-level diagram of 4×4 APUF is shown in Fig. 1.

1.3 PUF security

Reconfigurability of a PUF is very important in terms of security. A software clone of a PUF can be created given a sufficient amount of challenge-response pairs. Due to this, PUFs are divided into two broad categories: “weak” PUFs that accept only one or few different challenges and “strong” PUFs that accept a large amount of challenges, which makes them more secure.[2] 4×4 APUF implemented in this thesis is a strong PUF.

PUFs can be attacked by collecting and analyzing certain amount of challenge-response pairs. There are many ways to attack a PUF, such as “reverse engineering attack”, during which the architecture of the PUF is modeled to create a software clone that can later be used as the PUF itself to get responses to the challenges that have not been collected, and “collision attack”, during which identical responses of different PUFs are analyzed to guess other responses [17].

Delay-based PUFs are vulnerable to Machine Learning (ML)-based modeling attacks [20], categorized under reverse engineering attacks. 4×4 APUF implemented in this thesis is a delay-based PUF. During a modeling attack, paths in an APUF are modeled, the model is then trained with the help of ML algorithms using collected challenge-response pairs.

1.4 ChipWhisperer Tool-Chain

ChipWhisperer (CW) is a free tool-chain for side-channel power analysis and glitching attacks [5]. In this thesis, we used a target FPGA board on which our PUF is implemented, a capture board that is used to communicate with the FPGA board, and a software framework that is used to communicate with both target and capture boards, all provided by CW. We used this setup to collect challenge-response pairs from our PUF.

1.5 Related Work

The biggest weakness of PUF is that modeling attacks can break it given a sufficient amount of time and challenge-response pairs. The literature is full of

[2]The difficulty of breaking a PUF is inversely proportional to the percentage of challenge-response pairs that are revealed. So, security a PUF provides can be traded with its lifetime. Strong PUFs can be used with higher number of challenge-response pairs, which gives them a longer lifetime. In our case, a 28 stage 4×4 APUF accepts 24^{28} challenges, which makes infeasible to collect a sufficient amount of challenge-response pairs to easily brake its security.

novel PUF designs, mostly improvements to 2x2 APUF, aiming resistance to the modeling attacks. This is generally achieved either by increasing the complexity of the design to make the model difficult to build, and/or by increasing the number of parameters in the model to increase the required time and challenge-response pairs to brake the design. Many examples from the literature are presented in the following paragraphs.

Feed-Forward Arbiter PUF [16] FF-APUF improves 2x2 APUF by determining some of the challenges using the output of intermediary arbiters (feed-forward arbiters) put in between some of the switch blocks, rather than taking all the challenges from the user. Modeling attacks performed on regular 2x2 APUF do not work on FF-APUF and more sophisticated model is needed to imitate the new behavior.

Non-Linear Arbiter APUF [16] Improves 2x2 APUF by modifying the mapping from the challenge to switch block delays to make it non-linear. This modification makes the PUF resistant to physical probing attacks.

XOR-PUF [16] The response is produced by XORing responses from multiple APUFs. This multiplies the number of delays by the number of APUFs, thus, making modeling attacks more difficult.

Lightweight Secure PUF [18] Wraps multiple 2x2 APUFs by an "interconnect network", a logic circuit through which the challenge passes before being distributed to individual PUFs, "input networks" per PUF, through which challenges destined to individual PUFs pass after being processed by the interconnect network, and an "output network" outputs of all the PUFs pass to generate the final response. This complicates the modeling attacks.

Reconfigurable Optical PUF [15] Proposes "a structure that consists of a polymer containing randomly distributed light scattering particles". This structure produces a "steady" speckle pattern when exposed to a laser beam. The structure can be reconfigured by exposing it to a laser beam that is outside of operating conditions.

Phase Change Memory (PCM) based Reconfigurable PUF [15] PCM works by subjecting it to a specific heating pattern, which induces the resistivity of the material to change. High resistivity represents '1' and low resistivity '0'. Also, the intermediate states can be achieved with a writing operation that cannot be controlled; however, these states can be easily read. As a result, a long-lived random state can be created, which can be reconfigured at will.

The difference between "Reconfigurable Optical PUF" and "PCM based Reconfigurable PUF" compared to other PUFs presented in this section is that challenge-response behavior is uncontrollable. After reconfiguration, the previous challenge cannot be regenerated by another reconfiguration.

Logically Reconfigurable PUF [11] This is an implementation of a use case of PUFs: securely storing and reading a secret key. It combines a PUF with a non-volatile memory that stores state information. The state information is used to hash the response of the PUF and the hashing process can be reconfigured by changing the stored state.

Intrinsically reconfigurable D-RAM based PUF (D-PUF) [22] Normally, D-RAM based PUFs are "weak", thus, open to modeling attacks. This design proposes a "strong" D-RAM PUF. Manipulating the pausing time-interval during refresh operation of the memory changes the challenge-response behavior of the PUF. In other words, reconfiguration is achieved by changing the pausing interval.

R^{3}PUF [12] R^{3}PUF is based on memristive devices. It uses the resistance variations in memristive devices not only among CMOS devices but also among different reprogramming cycles within the same device. The response is generated by comparing two or more memristive-devices. The advantage of this design is that the responses are highly reliable (error-free).

Interpose PUF [19] Internally uses 2 XOR-PUFs, the upper layer and the lower layer. The challenge taken from the user is used as the challenge to the upper layer without modification, while the challenge to the lower layer is created by interposing the response of the upper layer into the challenge. This design makes modeling attacks more difficult while staying lightweight and strong (in the PUF sense).

MPUF [21] Multiplexer-based PUF (MPUF) uses responses of several APUFs as inputs and selectors of a multiplexer. The final output is the output of the multiplexer. This design aims to improve the vulnerability of the challenge against modeling and statistical attacks, and also its reliability.

Resistive RAM-based String Arbiter PUF [14] Proposes a string APUF based on a modified Resistive RAM. One key property of this design is that the APUF is realized within the memory array turning it to an APUF. Another key property is that it can be configured for different numbers of stages, which can be used to hide the number of bits in the challenge. This provides an additional layer of protection.

Majority Vote XOR-PUF [24] Solves the problem of XOR-PUF that is it cannot be realized with more than 12 internally used PUFs due to noise. Proposes majority voting for every PUF before XORing. This enables larger XOR-PUFs to be built, providing more resistant to ML attacks.

FF-XOR-PUF [2] FF-XOR-PUF is a combination of Feed-Forward Arbiter PUF and XOR-PUF, where each PUF in the XOR-PUF is an FF-PUF. Various versions with different kinds of FF-PUFs are proposed. This design aims to overcome the issues in XOR-PUF, such as, vulnerability to ML attacks and response instability.

FPGA implementation of a challenge pre-processing structure APUF [13] Proposes a pre-processing structure through which the challenge passes before going into the PUF. Each challenge bit passes through a modified version of RS-flip-flop, where the output of every flip-flop is acting at the same time as inputs of the adjacent flip-flops. This creates an additional behavior defined by the manufacturing differences, apart from the actual PUF itself. The aim is to make the design resistant to ML attacks.

2 Implementation

2.1 4×4 APUF

Our FPGA implementation for 4×4 APUF constitutes of several switch blocks, and an arbiter. The schematics of the hardware design is shown in Fig. 1.

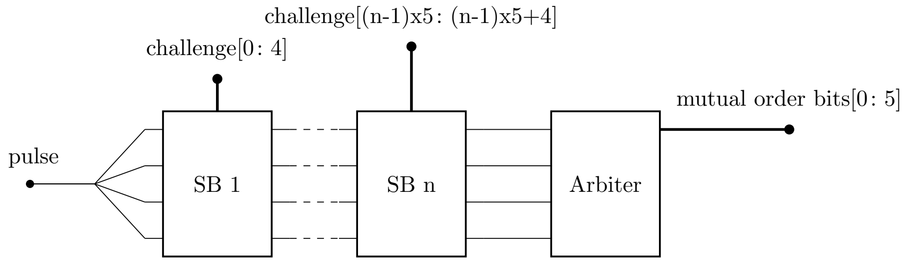


Figure 1: APUF high-level hardware design.

Every switch block has 4 inputs and 4 outputs. The inputs can be mapped the outputs in 24 (4!) different configurations specified by the 5-bit challenge representing numbers in the interval $[0, 23]$.

The arbiter compares every 6 pairs of the outputs belonging to the last switch block ($\binom{4}{2} = 6$), and generates a 6-bit response representing the order of the pulse for each pair.

In the rest of this thesis, we call an APUF with n switch blocks as an n-stage APUF.

2.1.1 Switch Blocks

The hardware design of a switch block is shown in Fig. 2.

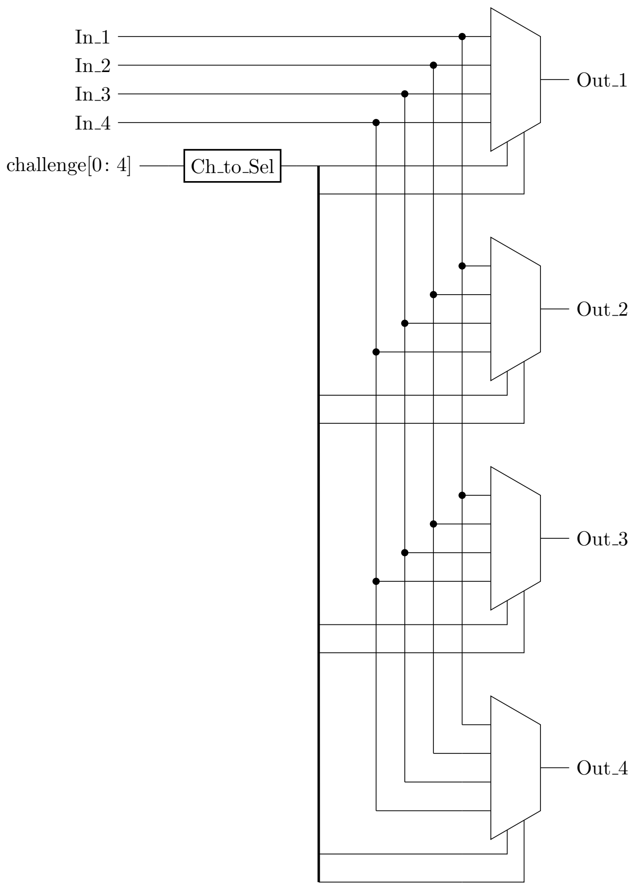


Figure 2: Switch block hardware design.

A switch block includes 4 4×1 multiplexers, each producing one of the 4 outputs. The 4-bit input of all the multiplexers are connected to the 4-bit input of the switch block. A challenge translation module is implemented to translate the 5-bit challenge that represents numbers in the interval $[0, 23]$, into 8 bits: 2-bit selectors for every multiplexer.

Equality of Paths In theory, in our implementation, the length of all 16 (4^2) paths in a switch block should be identical. In the ideal case, all these paths must be implemented symmetrically on the IC chip and the only factor causing the delays to differ must be the manufacturing differences. However, when it comes to implementing a design on an FPGA, there are many other factors that can cause the delays to differ; such as, the compiler decisions on the placement of the cells, the routing between these cells, the internal implementation of these cells and the look-up tables, etc. These additional factors can dominate the manufacturing differences and cause it to end up with predictable similarities between APUFs implemented on different IC chips.

In this thesis, we implemented and evaluated two different kinds of hardware placement for switch blocks and the arbiter. First one is the default placement performed by the design tool. In the second one, we manually placed all the switch blocks and the arbiter. The manual placement is performed to make the paths within and between switch blocks, and paths between the last switch block and the arbiter as symmetrical as possible.

Two switch blocks and routing between them are shown in Fig. 3 (default placement) and Fig. 4 (manual-placement).[3]

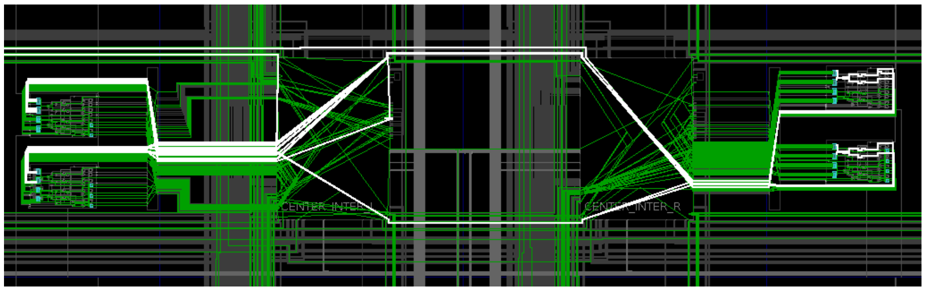


Figure 3: Two switch blocks with default placement. 4 6-LUTs belonging to the first switch block at right, 4 6-LUTs belonging to the second switch block at left.

[3]Only 4 out of 8 6-LUTs of a switch block, the ones belonging to the multiplexers, are manually placed.

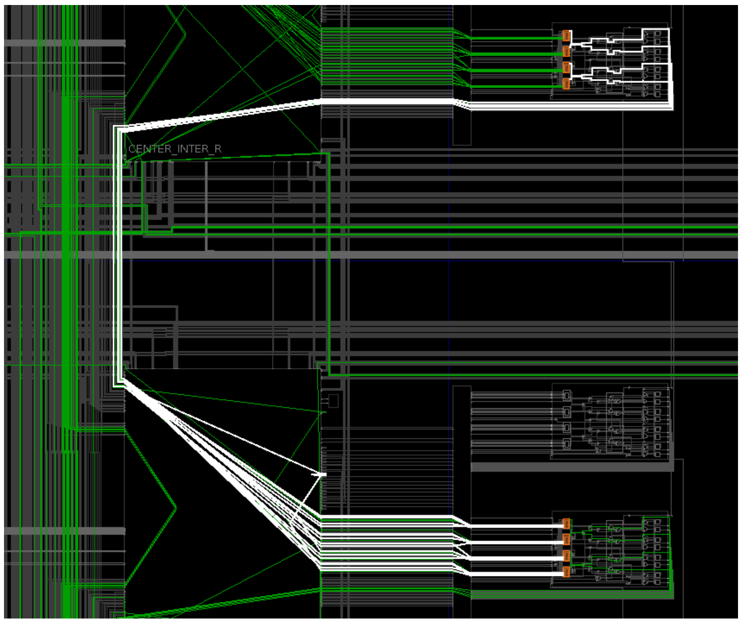


Figure 4: Two switch blocks with manual placement. 4 6-LUTs belonging to the first switch block on top, 4 6-LUTs belonging to the second switch block at the bottom.

The results, however, are better with the default placement. Accordingly, we only present results for the default placement.

Individual delays in the switch blocks are presented in the appendix.

2.1.2 Arbiter

The hardware design of an arbiter is shown in Fig. 5.

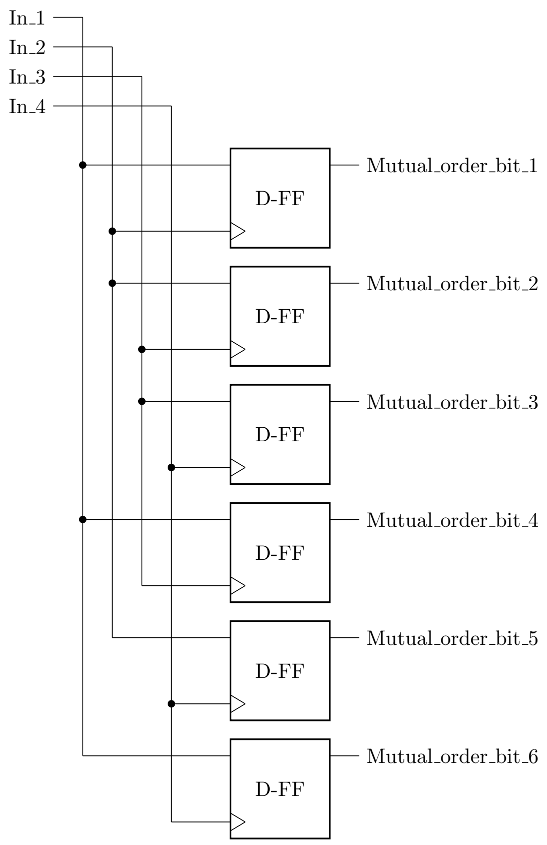


Figure 5: Arbiter hardware design.

The arbiter module includes 6 D-flip-flops. The D-input and the clock input of every one of them are connected to one of the input pairs it compares. If the path connected to the D-input is faster, the output is 1, otherwise, it is 0. In the rest of the thesis, the output of each of these every flip-flops is called a "mutual order bit".

The arbiter design proposed in the previous work [10] included the translation of the mutual order bits into a 5-bit number that represents one of the permutations of the 4 paths. We excluded this translation module and directly used mutual order bits.

The translation module proposed in [10] caused most of the illegal responses to be mapped to 0. This disturbed the response distribution. In other words, how the translation module was implemented influenced the outcome of statistical analysis performed in this thesis. To make the APUF responses independent from the implementation of the translation module, and to be able to detect illegal responses, the translation module is excluded. Illegal responses are explained in the rest of the thesis in detail.

Arbiter placement There is a fundamental difficulty at the placement and routing of arbiter flip-flops due to the fact that design tools handle clock paths differently: Clock paths and the components through which they pass within the cells are different from regular paths; furthermore, the clock signal is shared by all the flip-flops in a cell etc. During this thesis, we tried to place arbiter flip-flops manually, as we did with the switch blocks; however, since the default placement results were better, we present only them in this thesis.

2.2 APUF Driver

The process of getting a response from the APUF is as such: One should set the challenge to configure the switch blocks, then send a rising edge that is forked to all of the inputs of the first switch block, and finally read the mutual order bits from the arbiter. An APUF driver module is implemented as a Moore Finite State Machine (FSM) to perform this process. The hardware design of the driver is shown in Fig. 6. The state diagram of the FSM is shown in Fig. 7.

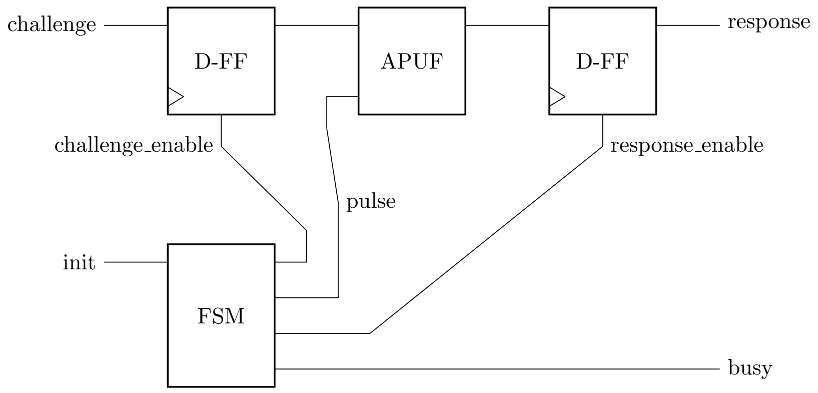


Figure 6: APUF driver hardware design.

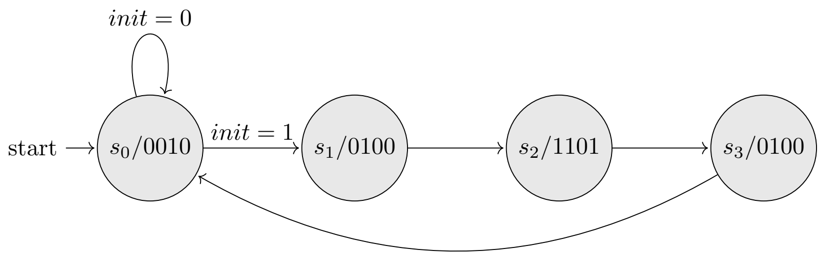


Figure 7: APUF driver Moore FSM state diagram. Output bits = (pulse, busy, challenge_enable, response_enable).

2.3 ChipWhisperer Wrapper

At the topmost level, the challenges are applied and responses are received by a computer program. The communication between the computer and the FPGA containing the APUF is performed via the CW tool-chain.

This wrapper contains 3 parts provided by CW: (1) a hardware design that wraps the APUF, (2) a set of circuit boards including the FPGA chip on which APUF is implemented, and (3) a software framework used to in the developed software that communicates with the APUF.

The tool-chain encapsulates the design unit implemented on the FPGA, abstracts the inputs to the unit as the “plaintext” and the “key”, and abstract the output from the unit as the “ciphertext”. In our case plaintext is the combined challenge to the switch blocks. It is $5n$ bits in size where n is the number of stages. The ciphertext is the response taken from the arbiter (mutual order bits), 6 bits in size. Since the key is the FPGA chip itself, that input is not in use.

Since CW offers 128 bits for the plaintext and the ciphertext, some of the spare bits are used to get a predefined signature as part of the response for debugging purposes.

In our thesis, CW is only used to apply challenges to the APUF and receive responses in an easy way. Even though performing other features of the tool-chain, such as power trace capturing and correlation power analysis, is not in the scope of this thesis, they are among the possible future work. So, using this tool-chain is also a preparation for future research.

Once challenge-response pairs are received using CW, all the data analysis software is written independently from the tool-chain, without needing the CW framework.

2.4 Testbed

- FPGA target board: 2 distinct CW305 Artix FPGA Target boards [7] with Xilinx Artix-7 XC7A100T FPGA [1]
- Capture board: CW1173 ChipWhisperer-Lite [6]
- Hardware design tool: Xilinx Vivado v2019.2.1 (64-bit) [23]
- Computer operating system: Ubuntu 18.04.4 LTS
- Software: ChipWhisperer 5.0 [4]

A photo of the target FPGA board and the capture board together is shown in Fig. 8. Other ends of the two USB cables connected to each board are connected to the computer.

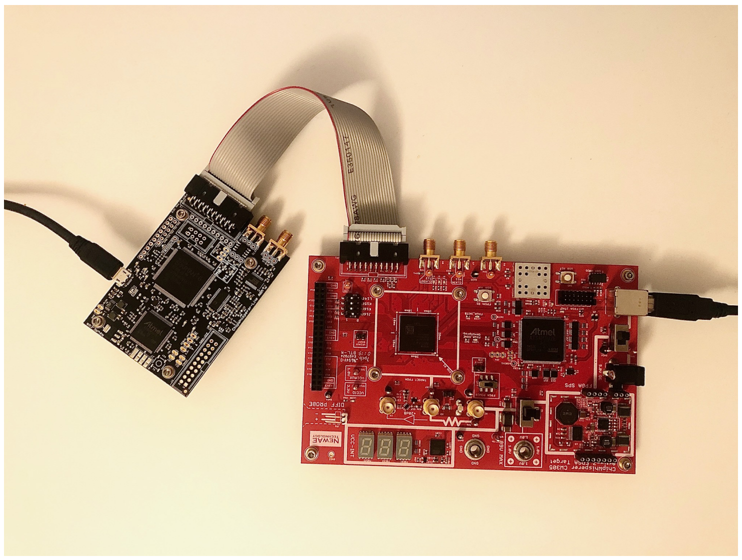


Figure 8: CW305 Artix FPGA Target board (right) and CW1173 ChipWhisperer-Lite capture board (left).

2.5 Further Details on Implementation

This particular implementation of APUF is not intended to be resistant to hardware attacks, such as power and side-channel analysis, but sufficient to evaluate the statistical properties of the proposed APUF implemented on FPGAs.

We developed our APUF hardware design in VHDL, with "number of stages" as a generic input.

Since an APUF design does not make sense in the digital level but in the analog level, the design compiler corrupts the design during optimizations. To prevent this, we disabled compiler optimizations for switch blocks and the arbiter. This was done via "DONT_TOUCH" compiler attribute of Vivado.

During the development of the APUF hardware design, alongside our main development board, it is also loaded on Nexys 4 DDR FPGA board and controlled via switches and LEDs.

3 4×4 APUF Model

We propose a new mathematical model for 4×4 APUF using homogeneous coordinates.

The behavior of a switch block can be expressed as two consecutive operations with 4 inputs and 4 outputs: "permutation" that changes the order of the inputs and "translation" that adds certain delays to the input.

We will express the input and the output, $\boldsymbol{i} = [i_1, i_2, i_3, i_4]^T$ and $\boldsymbol{o} = [o_1, o_2, o_3, o_4]^T$, in homogeneous coordinates as $\boldsymbol{i} = [i_1, i_2, i_3, i_4, 1]^T$ and $\boldsymbol{o} =$

$[o_1, o_2, o_3, o_4, 1]^T$.

This will enable the translation operation to be expressed with a matrix multiplication.

Permutation This operation can be performed by multiplying the input with the matrix $\boldsymbol{P}$, created using 4 distinct standard basis vectors put in the desired permutation. An example of a permutation operation that changes the order of the first and the second inputs is as follows:

$$\underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{P}} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ 1 \end{bmatrix} = \begin{bmatrix} i_2 \\ i_1 \\ i_3 \\ i_4 \\ 1 \end{bmatrix}$$

The bold part of $\boldsymbol{P}$ shows the standard basis vectors.

Translation This can be achieved by multiplying the input with the matrix $\boldsymbol{T}$ as follows:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \mathbf{\Delta d_1} \\ 0 & 1 & 0 & 0 & \mathbf{\Delta d_2} \\ 0 & 0 & 1 & 0 & \mathbf{\Delta d_3} \\ 0 & 0 & 0 & 1 & \mathbf{\Delta d_4} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{T}} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ 1 \end{bmatrix} = \begin{bmatrix} i_1 + \Delta d_1 \\ i_2 + \Delta d_2 \\ i_3 + \Delta d_3 \\ i_4 + \Delta d_4 \\ 1 \end{bmatrix}$$

The combined operation for permutation and translation can be expressed with matrix $\boldsymbol{S}$, which represents the compete behavior of a switch block. Corresponding matrix $\boldsymbol{S}$ created with above $\boldsymbol{P}$ and $\boldsymbol{T}$ is in the following form:

$$\boldsymbol{S} = \boldsymbol{TP} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{\Delta d_1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{\Delta d_2} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{\Delta d_3} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{\Delta d_4} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Delay values $\Delta d_1, \ldots, \Delta d_4$, and the order of the bases vectors are functions of the challenge of the switch block. These functions can be expressed using 16 distinct delay parameters belonging to 16 paths.

The combined behavior of N switch blocks can be expressed by multiplying each matrix $\boldsymbol{S}$, belonging to every switch block. The resulting matrix is in the following form:

$$\prod_{n=1}^{N} \boldsymbol{S}_n = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} & D_1 \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} & D_2 \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} & D_3 \\ p_{4,1} & p_{4,2} & p_{4,3} & p_{4,4} & D_4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where $p_{m,n}$ are elements in the combined permutation matrix and constitutes of distinct standard basis vectors and $D_1, \ldots, D_4$ are combined delay values.

The input of the first switch block, the stimuli, can be defined as $\boldsymbol{s} = [0, 0, 0, 0, 1]^T$ since there is no accumulated delay. Then, the equation that binds the stimuli to the output of the last switch block is as follows:

$$\begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} & D_1 \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} & D_2 \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} & D_3 \\ p_{4,1} & p_{4,2} & p_{4,3} & p_{4,4} & D_4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ 1 \end{bmatrix}$$

The model The final delays $D_1, \ldots, D_4$ going to the arbiter can be expressed as follows:

$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ 1 \end{bmatrix} = \left(\prod_{n=1}^{N} \boldsymbol{S}_n \right) \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

This model can be applied to APUFs with switch blocks of arbitrary size easily by changing the matrix sizes.

Extension for arbiter paths The influence of the paths from the last switch block to the arbiter flip-flops can be added to the model with a matrix $\boldsymbol{F}$ that transforms the 4 delays $D_1, \ldots, D_4$ into 6 delay differences $\Delta D_{1,2}, \Delta D_{2,3}, \Delta D_{3,4}, \Delta D_{1,3}, \Delta D_{2,4}, \Delta D_{1,4}$ going to the flip-flops as follows:

$$\begin{bmatrix} \Delta D_{1,2} \\ \Delta D_{2,3} \\ \Delta D_{3,4} \\ \Delta D_{1,3} \\ \Delta D_{2,4} \\ \Delta D_{1,4} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & e_{1,2} \\ 0 & 1 & -1 & 0 & e_{2,3} \\ 0 & 0 & 1 & -1 & e_{3,4} \\ 1 & 0 & -1 & 0 & e_{1,3} \\ 0 & 1 & 0 & -1 & e_{2,4} \\ 1 & 0 & 0 & -1 & e_{1,4} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{F}} \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ 1 \end{bmatrix}$$

where $e_{m,n}$ is error introduced by the flip-flop that compares paths m and n. The extended model is as follows:

$$\begin{bmatrix} \Delta D_{1,2} \\ \Delta D_{2,3} \\ \Delta D_{3,4} \\ \Delta D_{1,3} \\ \Delta D_{2,4} \\ \Delta D_{1,4} \\ 1 \end{bmatrix} = \boldsymbol{F} \left(\prod_{n=1}^{N} \boldsymbol{S}_n \right) \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

4 Data Collection & Analysis

Data is collected as challenge-response pairs. One pair consists of the response an APUF generates for a particular challenge. A predefined set of challenges was applied to different APUF setups. In this thesis, the process of applying the set of challenges for a particular setup is called an "experiment". The parameters defining a setup are (1) number of stages of the APUF, (2) the unique FPGA chip on which APUF is implemented. Also each experiment is performed multiple times to perform reliability analysis.

1, 2, 3, 4, and 24 stages are used in the experiments. All possible challenges were applied to APUFs with 1, 2, 3, and 4 number of stages, while a randomly chosen set of challenges was applied to a 24 stage APUF since considering a challenge set of such large size is not realizable. All these experiments were repeated for two different FPGA chips.

24 stage APUF is the main focus during data collection and analysis. CW only supports 128-bit plaintext that is used to communicate the challenge. The largest number of stages having a challenge that can fit into 128 bits is 24 (120-bit challenge). This selection was to keep the communication with the FPGA simple, at the same time, having large enough number of challenges to make brute force attacks infeasible.

4.1 Format of Responses

All of the analysis is performed on individual responses, rather than a concatenation of multiple responses. In a real-world application, it may be necessary to combine multiple responses of an APUF to achieve a larger response (such as a 128-bit response) for utility purposes.

Most of the analyses were performed on 6 mutual order bits. For some of the analysis, we translated the mutual order bits into an integer in the interval $[0, 23]$ representing a particular permutation of incoming signals to the arbiter.[4]

4.2 Influence of Arbiter Paths

During our experiments, an anomaly in the responses was discovered: Delay differences within the path pairs, leading from the last switch block to each flip-flop in the arbiter, cause some of the mutual order bits to be incorrect. This

[4]The translation is performed on a computer.

sometimes results in responses that cannot be translated into the permutation information. These responses are called “illegal” in this thesis. And others are called “legal”.

This concept can be demonstrated with an example: Let response bits (a, b, c, d, e, f) represent the comparison between output pairs of the last switch block, respectively, (o_1, o_2), (o_2, o_3), (o_3, o_4), (o_1, o_3), (o_2, o_4), and (o_1, o_4). If p_1, the path leading to o_1, is faster than p_2, and p_2 is faster than p_3, then, p_1 is concluded to be faster than p_3. So, responses, where $a = b \neq d$, are illegal. Yet, during experiments some of the responses we got were illegal.

This phenomenon was not analyzed in [8] and [10]. The mathematical model proposed in [10] only includes delays of the paths within the switch blocks. The model provided in this thesis considers flip-flop paths, as well as switch block paths.[5]

4.2.1 Response Correction

Some of the illegal responses can be corrected as such: Number of legal responses is 24 (4! is the number of possible permutations of 4 paths), and the number of illegal responses is 40 ($64 = 2^6$ in total). 24 of the illegal responses have only one legal response that is 1 Hamming distance away, while 16 of them have 3 legal responses that are 1 Hamming distance away. Under the assumption that the probability of paths leading to the arbiter causing multiple bit changes is lower enough than only a single bit change, 24 of illegal responses can be corrected.

4.2.2 Analysis of Illegal Responses

Illegal responses were analyzed by looking at the distribution of all 40 illegal responses, their Hamming weight distribution, and most importantly transition (transition from legal to illegal) of individual bits during response correction. Transition information helps to spot the problematic bits in the response.

4.3 Uniformity Analysis

In uniformity analysis, the distribution of responses of one particular APUF to a set of random challenges is evaluated. It is performed intra-APUF. We performed it on data collected from one experiment consisting of a 24 stage APUF implemented on one particular FPGA chip. The challenge set in the experiment consists of 13824 challenges picked randomly in a uniform way.

We looked at the average Hamming weights (distribution of 1s and 0s) in each 6 mutual order bits, and also overall. In the theoretical case, where responses are perfectly uniform, average Hamming weights of each mutual order bit should be 0.5, and the overall Hamming weight of responses should be 3.

The rest of the uniformity analysis was performed 2 times; first, by ignoring the illegal responses; second, by correcting them.

[5]These additional delays can also cause a legal response to flip to another legal response.

We translated mutual order bits into integers in the interval $[0, 23]$, and looked at their distribution.

We also looked at the Hamming weight distribution of the permutation order after dividing it by 3. This is done in [10] inside arbiter to directly achieve uniform bit distribution. After division, the interval $[0, 23]$ is reduced to $[0, 7]$, which uses all the bits uniformly when represented in binary with 3 bits.

4.4 Reliability Analysis

In theory, an ideal APUF should generate the same responses when the same challenge is applied over and over again. However, this is not the case in the real world. Our experiments show that APUF responses are nondeterministic. In other words, the application of the same challenge over and over again does not always produce the same response.

In this thesis, we define the reliability of a challenge to whether it always generates the same response or not. Challenges that always generate the same response are called "reliable challenges", others "unreliable challenges".

Reliability analysis is performed to analyze the reliabilities of every challenge for one particular APUF. It is performed intra-APUF. We ran one experiment 407 times on a 24 stage APUF implemented on one particular FPGA chip. The challenge set in the experiment consists of 13824 challenges picked randomly in a uniform way (the same set used in uniformity analysis).

We looked at the distribution of the responses per every challenge across multiple applications of the same experiment. In the ideal case, where all the challenges are reliable, every challenge generates one single response. So, the ideal distribution is $(13824, 0, 0, \dots)$, where there are 407 entries. The n^{th} entry is the number of challenges that generate n different responses over 407 repeated experiments, $n = 1, 2, \dots, 407$. The first entry is the number of challenges that generate one single response during all 407 experiments. Its percentage over the sum of all the entries gives the percentage of reliable challenges.

4.5 Uniqueness Analysis

The essence of APUF is that it should produce different results when implemented on different chips. In other words, responses to a particular set of challenges should be unique for every chip. Uniqueness analysis analyzes the degree of uniqueness. It is performed inter-APUF.

Uniqueness analysis is performed by running one experiment 407 times on 2 different FPGA chips on which a 24 stage APUF is implemented. The challenge set in the experiment consists of 13824 challenges picked randomly in a uniform way (the same set used in uniformity analysis). Later on, analyses are performed on every 407 pairs of experiments (2 experiments for different chips in a pair). Finally, the results for every pair are averaged.

We are defining the theoretically ideal case as such: responses are perfectly random, illegal responses are assumed not to occur, and responses are perfectly reliable. In this ideal case, the probability for each legal response to occur is $\frac{1}{24}$.

The probability of responses to the same challenge from 2 APUFs implemented on different chips to be different, also the expected number of challenges that produce different results on 2 different chips is $\frac{23}{24} = 95.8333\%$. During the analysis, the actual number of challenges that produced different results on 2 different chips are compared with this ideal value.

There is one complication due to unreliable challenges. When responses from different chips differ, it is difficult to understand whether the cause is unreliability or manufacturing differences. Furthermore, as our experiments show, unreliable challenge sets are highly different for every chip. As a solution we also calculate uniqueness by excluding the union of unreliable challenges.

5 Results

Results are for 24 stage 4×4 APUF unless stated otherwise.

5.1 Uniformity Analysis Results

The number of challenges in the experiment was 13758 and 66 of them were illegal. The percentage of legal responses is 99.52%.

Among 66 illegal responses, none was corrected.

Following results are calculated by ignoring the presence of illegal responses, because the influence of illegal response is negligible.

Response distribution is shown in Fig. 9. The Hamming weight distribution of responses divided by 3 is shown in Fig. 10.

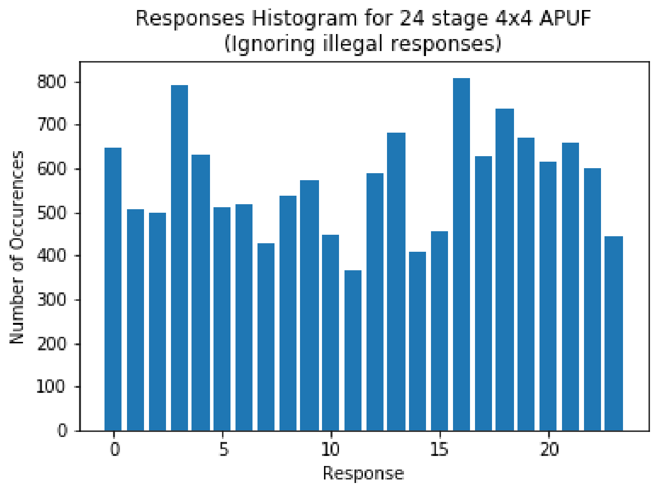


Figure 9: Response histogram for 24 stage 4×4 APUF (ignoring illegal responses).

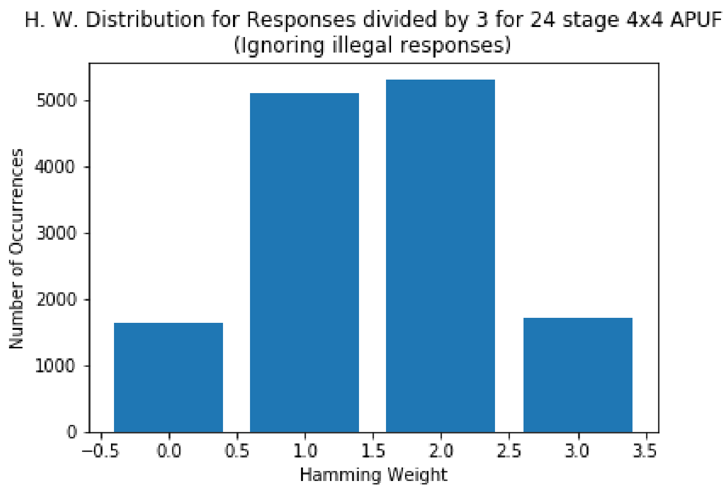


Figure 10: Hamming weight distribution for responses divided by 3 for 24 stage 4×4 APUF (ignoring illegal responses).

5.1.1 Distribution of Mutual Order Bits

Average of every mutual order bit is shown in Table 1.

Bit	1	2	3	4	5	6
Aver.	0.5579	0.4862	0.5207	0.5097	0.4331	0.4901

Table 1: Average of every mutual order bit.

The sum of these numbers, in other words, the average Hamming weight of mutual order bits is 2.9978.

The Hamming weight distribution of legal responses (in mutual order bits format) is shown in Fig. 11.

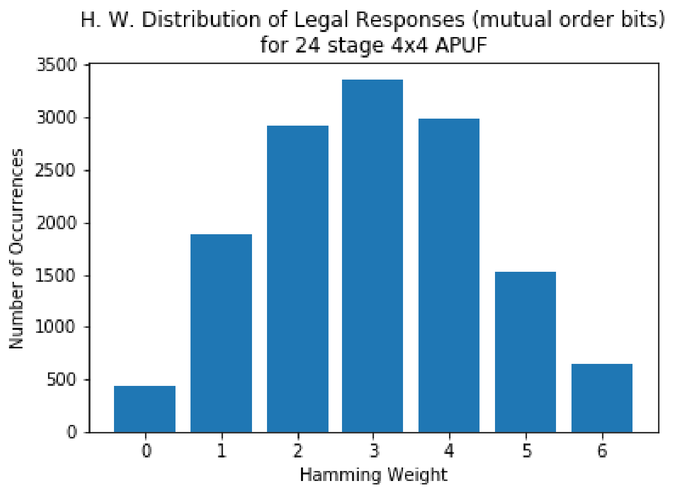


Figure 11: Hamming weight distribution of legal responses (mutual order bits) for 24 stage 4×4 APUF.

5.2 Reliability Analysis Results

At the result of repeated applications of the same experiment, the distribution of the number of different responses for 2 FPGA chips is shown in Table 2.

# of responses	1	2	3	4	5	6	...
Chip 1	12835	957	22	10	0	0	...
Chip 2	12714	1063	30	16	1	0	...

Table 2: The distribution of the number of different responses for 2 FPGA chips.

The percentage of the challenges that produced only a single response throughout the repeated experiments (first entry of Table 2 divided by the sum of all the entries), in other words, the percentage of reliable challenges is shown in Table 3.

Chip 1	92.8458%
Chip 2	91.9704%

Table 3: The percentage of reliable challenges.

The probabilities of a randomly selected challenge to produce its n^{th} likely response are shown in Table 4, $n = 1, 2, \ldots$.

Responses	1st	2nd	3rd	4th	5th	...
Chip 1	99.0599%	0.9297%	0.0087%	0.0017%	0%	...
Chip 2	98.8828%	1.1015%	0.0133%	0.0023%	0%	...

Table 4: The probabilities of a randomly selected challenge to produce its n^{th} likely response, $n = 1, 2, \ldots$.

On average for all chips, the probability of a randomly selected challenge to produce its most likely response is 98.9714%. These probabilities are calculated by analyzing response histograms for individual challenges.

5.3 Uniqueness Analysis Results

Percentage of unreliable challenges (calculated using results of reliability analysis) for every FPGA chip is shown in Table 5.

Chip 1	7.1542%
Chip 2	8.0295%

Table 5: The percentage of unreliable challenges.

Percentage of the union of unreliable challenges for 2 chips: 13.5923%

The percentage of challenges that produced different results on 2 different chips (illegal challenges included) is 15.1520%.

The same percentage when the union of unreliable challenges from 2 chips are excluded is 9.3261%.

5.4 Illegal Response Analysis

The distribution of illegal responses is shown in Fig. 12. Every 40 slot on the x-axis represents one of the illegal responses.

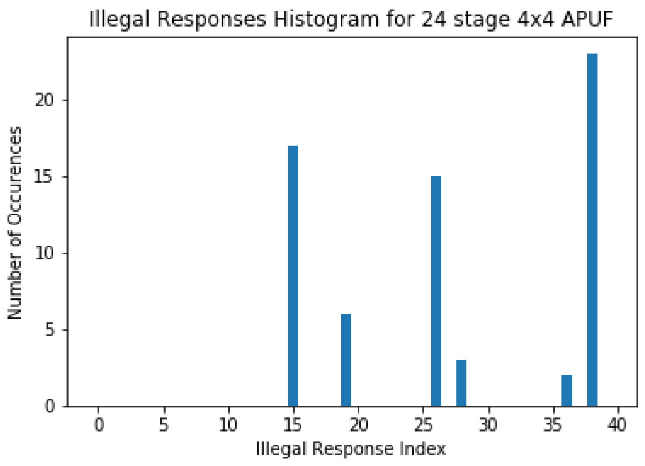


Figure 12: Illegal responses histogram for 24 stage 4×4 APUF.

The Hamming weight distribution of illegal responses (in mutual order bits format) is shown in Fig. 13.

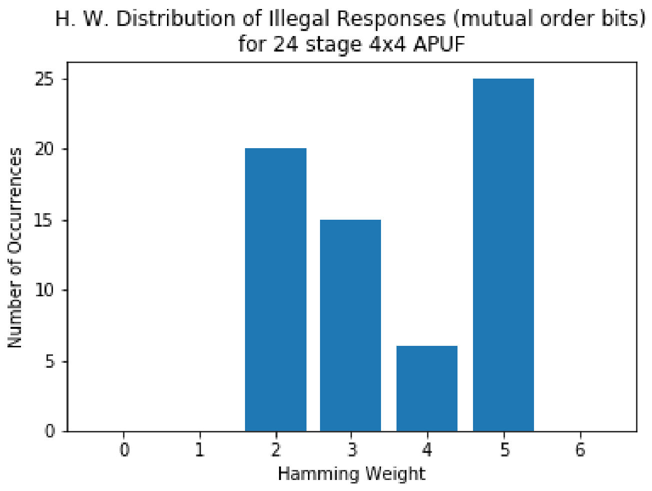


Figure 13: Hamming weight distribution of illegal responses (mutual order bits) for 24 stage 4×4 APUF.

Interesting results from 4 stage 4×4 APUF When a random set of challenges are applied to our 4 state APUF, the percentage of illegal responses were much higher: 24.6672%.

This is supposed to be the result of paths between the last switch block and the arbiter flip-flops. In that particular implementation, because of design compiler decisions, look-up table locations, routing, etc., the difference between some pairs of paths to the flip-flops must have turned out to be high, which caused some mutual order bits to stuck at a value for a large number of challenges.

This result is shown in Fig. 14 that shows bitwise transitions of legal responses to illegal responses. The problematic flip-flops are number 0 and 1. This plot is created by correcting 3410 out of 10414 illegal responses.

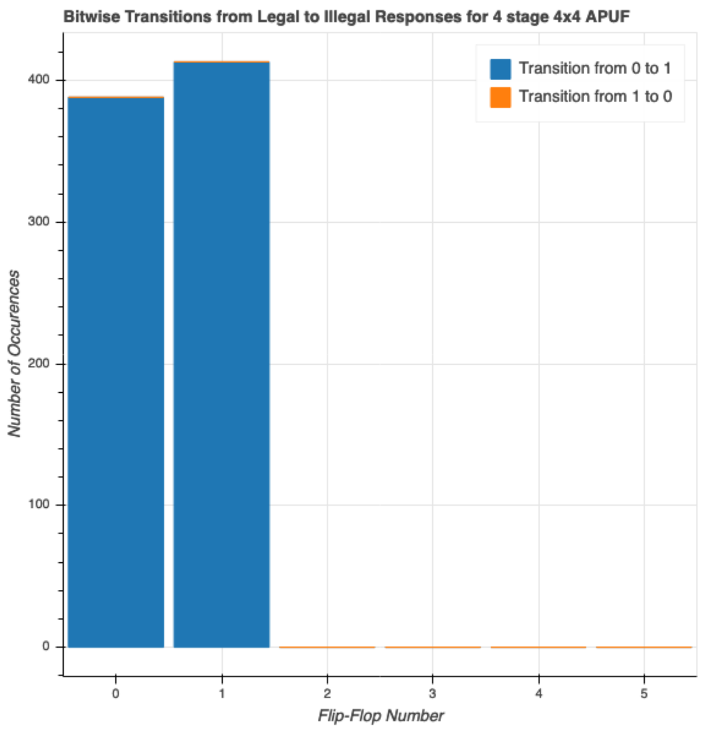


Figure 14: Bitwise transitions from legal to illegal responses for 4 stage 4×4 APUF.

Although not presented in this thesis, all other analyses for 4 stage APUF are indicating this behavior.

5.5 Area Comparison

The area comparison of 4×4 APUF, implemented in this thesis, to different variants of 2x2 APUFs is presented in Table 6. Table columns are, respectively, APUF type, number of stages, challenge length in bits, response length in bits, number of 6 Input Loop-Up Tables (6-LUTs), number of flip-flops, number of 6-LUTs per one bit of response, number of flip-flops per 1 bit of response.

APUF type	Stages	Ch. bits	Res. len.	6-LUT	FF	6-LUT/res. len	FF/res. len
2x2 APUF [16]	128	128	1	256	1	256	1
8-XOR APUF [16]	128	128	1	2050	8	2050	8
Interpose APUF [19]	128	128	1	514	2	514	2
CRC-APUF [9]	128	128	1	320	1	320	1
4×4 APUF	28	140	4.58	224	6	48.86	1.31

Table 6: Area comparison of 4×4 APUF to other various 2x2 APUF variants.

The number of stages of 4×4 APUF is selected as 28 so that the number of possible challenges is equal or greater than 2^{128}.[6]

Compared to others, 4×4 APUF generates 5-bit response that represents a number in the range $[0, 23]$, rather than a single bit in the range $[0, 1]$. Since the interval $[0, 23]$ cannot be represented with an integer number of bits without redundancy, we are calculating the response length of 4×4 APUF as $\log_2 24 \cong 4.58$. This is an advantage because to achieve the same amount of output, other APUFs should be run more than once or more than one instance of them should be implemented in parallel. It should also be underlined that critical path (path from the input stimuli, through all of the switch block, to the arbiter) of 4×4 APUF is shorter than 2x2 APUF, enabling response generation with a higher throughput.

6 Discussion

Repeating the research question: Is APUF with 4×4 switch blocks realizable on FPGAs with a sufficient amount of exploitation of manufacturing differences to be used in real-world applications?

First of all, our 4×4 APUF design extracts some amount of manufacturing differences. This means our design can be useful for some real-world applications.

At first look, our design seems to have low uniqueness. Nevertheless, it should be kept in mind that our results are for responses of small length (4.58 bits). The overall uniqueness can be improved by combining multiple responses to create a larger one. Though, while doing so, non-ideal reliability should be taken into account.

6.1 Our 4×4 APUF in the Real World

We are proposing several methods to make better use of our 4×4 APUF and applications to which these methods can be applied.

In all of these methods multiple responses are "combined" to produce a larger one. We are defining this combination as such: Let R be the combination of $r_1, \dots r_m$, individual responses taken from the APUF.

[6]Keeping the number of challenge bits just above 128 would be wrong because challenge bits are redundant: 5-bit challenge of a switch block can only take values in the interval $[0, 23]$, rather than $[0, 31]$.

$$R = 24^0 r_1 + \cdots + 24^{m-1} r_m$$

We are performing the combination this way to prevent redundancy in the combined response.

Combining Responses of Randomly Selected Challenges This is the most straightforward method. According to our results, the probability of a randomly selected challenge to produce different results on 2 different chips is 15.1520%. Let's call it p. Accordingly, the probability, u, of 2 different combined responses created from 2 different chips to be different can be expressed as follows.

$$u = 1 - (1-p)^m \tag{1}$$

where m is the number of responses that are combined.

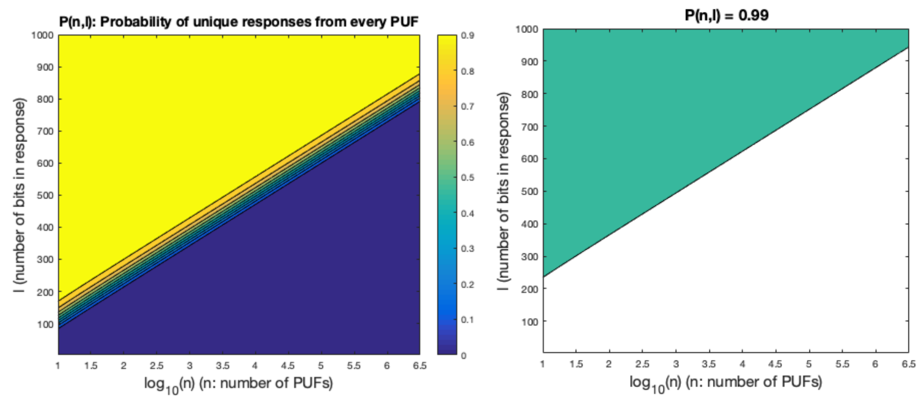
Let's generalize the case and call the same probability for n different chips $P(n)$, where $P(2) = u$. If we assume that number of PUFs is negligible compared to the possible number of combined responses ($n \ll 24^m$), $P(n)$ can be expressed as follows.

$$P(n) = u^{\binom{n}{2}} = u^{\frac{n(n-1)}{2}}, \quad n \geq 2 \tag{2}$$

If we combine equations 1 and 2, we can express P as a function of both n and m.

$$P(n,m) = \left[1 - (1-p)^m\right]^{\frac{n(n-1)}{2}} \tag{3}$$

A contour diagram of P is shown in Fig. 15. The sufficient number of response length for the required number of PUFs can be seen on the diagram. For example, if we want to uniquely identify 10000 PUFs with 99% probability, we need a 600-bit response.



(a) General

(b) Only the contour where $P = 0.99$

Figure 15: Contour diagram for P (uniqueness probabilities).

The problem with this method is that unreliability of responses is not handled. If the same combined response is tried to be generated using the same challenges, the results will most probably differ, especially for higher m values. However, this unpredictability can be exploited by applications where True Random Number Generators (TRNGs) are necessary.

Detection of Reliable Challenges Beforehand This method involves using only reliable challenges by detecting them beforehand. Detection can be performed either after fabrication before distribution or while runtime. This way, it will be possible to reproduce the same responses. This enables identification and authentication applications.

Equation 3 can also be used for this method with, this time, $p = 9.3261\%$. Need for larger responses (higher m) and extra cost for reliable challenge detection are traded with repeatability.

Better uniqueness would improve usage costs of our 4×4 APUF design by reducing the number of required responses to combine. However, there is a possibility that higher uniqueness can also cause lower reliability because when PUF delays are closer to each other, the results will be more sensitive to factors like noise.

7 Future Work

First of all, the results of our 4×4 APUF do not have desirable uniqueness. So, the hardware design should be improved to achieve higher uniqueness. This may be achieved by focusing on symmetrical placement and routing of switch blocks and the arbiter.

Once an acceptable uniqueness is achieved, the security of the 4×4 APUF should be evaluated. Evaluating its resistance to ML modeling attacks has the priority since APUFs are generally vulnerable to these attacks. Resistance to side-channel analysis can also be evaluated.

Real-world applications require at least 128-bit keys. Accordingly, APUF responses should be securely transformed into desired length. Also statistical analysis of this transformation should be performed.

In this thesis, we provided an area comparison of our 4×4 APUF to other APUFs in the literature. This comparison can be extended. After security analysis, the size of different APUFs (like the number of stages) can be selected to provide a similar amount of security, then areas can be compared. This way, area requirement can be related to security every APUF provides. Furthermore, the maximum response generation throughput should be determined experimentally and play a role in the comparison.

8 Conclusion

In this thesis, we implemented 4×4 APUF proposed in [8] on FPGAs and performed statistical analysis to evaluate its capabilities to be used in real-world applications.

According to the analysis, the capabilities of our design are adequate to enable many real-world applications, such as identification, authentication, encryption, and random number generation. However, since uniqueness is lower than desirable, these applications are associated with runtime performance penalty.

Our design can be improved by focusing on making switch block delays closer to each other, which may increase uniqueness. Other possible future work includes performing security evaluation of 4×4 APUF, most importantly evaluating its resistance to ML modeling attacks, and finding better methods that enable the usage of 4×4 APUF with lower operation costs.

Appendices

A Hardware Schematics for 4×4 APUF

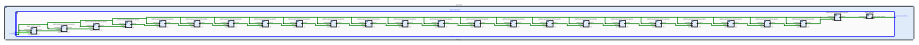


Figure 16: Vivado schematics for 4×4 APUF.

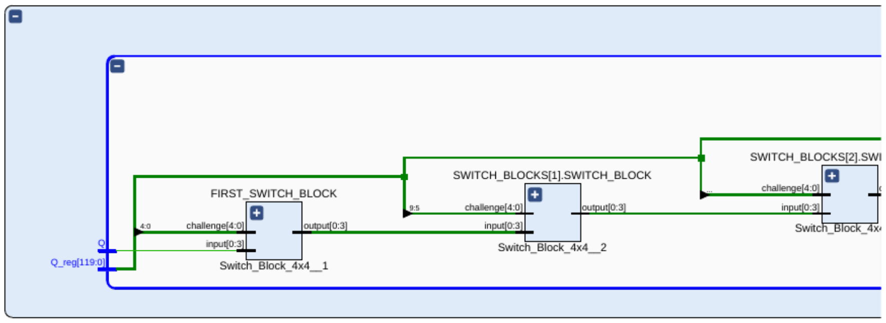


Figure 17: Vivado schematics for 4×4 APUF (Front closeup).

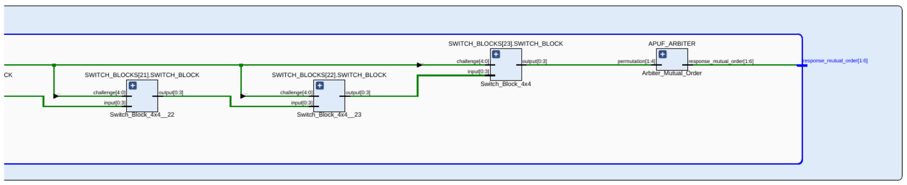

SWITCH_BLOCKS[21].SWITCH_BLOCK
Switch_Block_4x4__22
SWITCH_BLOCKS[22].SWITCH_BLOCK
Switch_Block_4x4__23
SWITCH_BLOCKS[23].SWITCH_BLOCK
Switch_Block_4x4
APUF_ARBITER
permutation[1:4]
response_mutual_order[1:6]
Arbiter_Mutual_Order

Figure 18: Vivado schematics for 4×4 APUF (Back closeup).

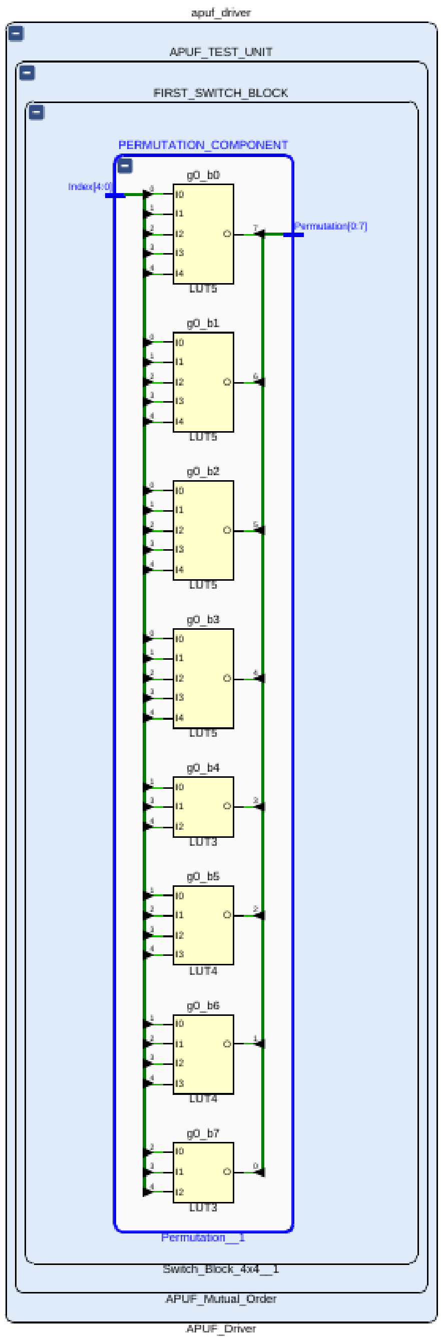


Figure 19: Vivado schematic for 4×4 APUF permutation component.

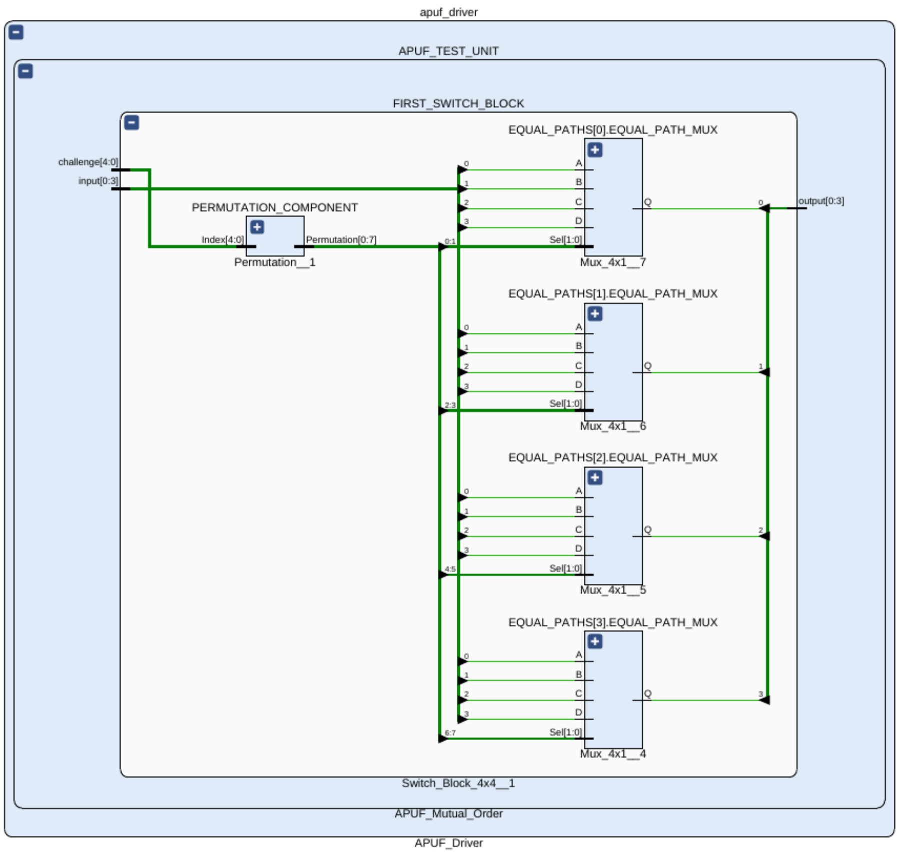


Figure 20: Vivado schematics for 4×4 APUF switch block.

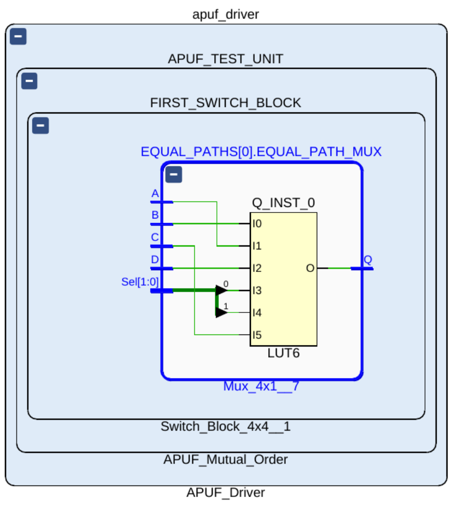


Figure 21: Vivado schematics for 4×4 APUF switch block multiplexer.

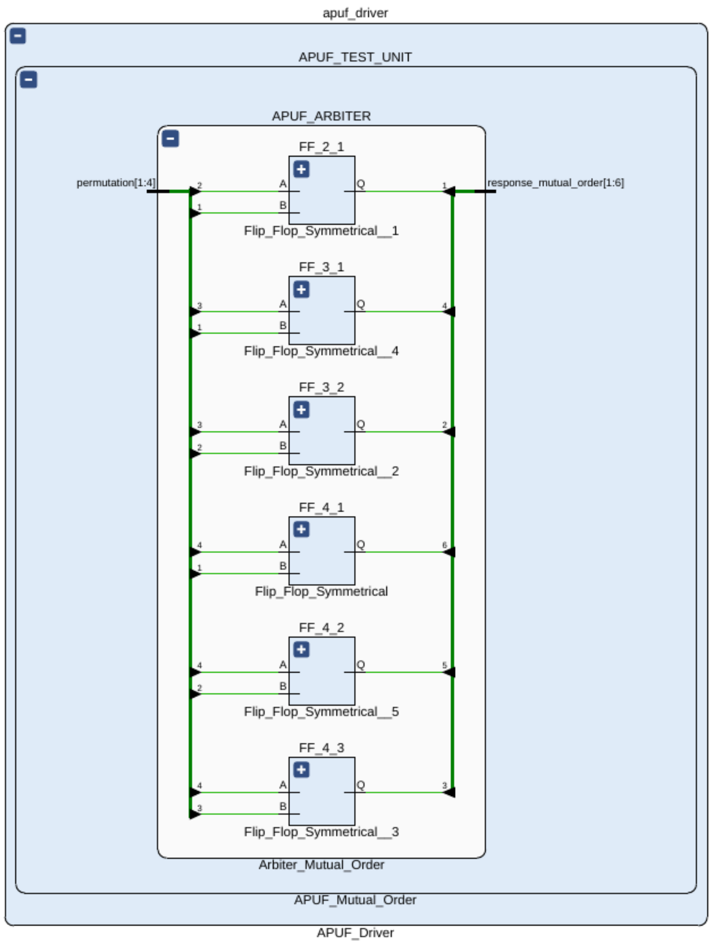


Figure 22: Vivado schematics for 4×4 APUF arbiter.

B 4 × 4 APUF Net Delays

The output of the Tcl script written to get net delays in Vivado. Using default placement.

Manually replaced repetitive parts of the net names with three dots “...”.

```
source ~/Downloads/get_net_delays.tcl
# puts "NAME FAST_MAX FAST_MIN SLOW_MAX SLOW_MIN"
NAME FAST_MAX FAST_MIN SLOW_MAX SLOW_MIN
# proc put_delays {net} {
#     foreach net_delay [get_net_delays -of_objects [get_nets $net]] {
#  set fast_max [get_property FAST_MAX $net_delay]
#  set fast_min [get_property FAST_MIN $net_delay]
#  set slow_max [get_property SLOW_MAX $net_delay]
#  set slow_min [get_property SLOW_MIN $net_delay]
#  puts "$net_delay $fast_max $fast_min $slow_max $slow_min"
#     }
# }
# proc switch_block_delays {net_base} {
#     for {set idx 0} {$idx < 4} {incr idx} {
#  put_delays $net_base[$idx]
#     }
# }
# switch_block_delays apuf_driver/APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/input
...FIRST_SWITCH_BLOCK/input[0]_to_...apuf_response_mutual_order_reg_reg[1]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[0]_to_...apuf_response_mutual_order_reg_reg[2]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[0]_to_...apuf_response_mutual_order_reg_reg[3]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[0]_to_...apuf_response_mutual_order_reg_reg[4]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[0]_to_...apuf_response_mutual_order_reg_reg[5]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[0]_to_...apuf_response_mutual_order_reg_reg[6]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[0]_to_...FSM_onehot_state_reg[3]/D 611 524 1123 944
...FIRST_SWITCH_BLOCK/input[0]_to_...FSM_onehot_state[0]_i_1/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[0]_to_...busy_INST_0/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 716 610 1304 1089
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 712 606 1303 1087
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 961 822 1757 1471
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 689 594 1252 1054
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 770 665 1407 1185
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 867 761 1492 1273
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 963 839 1688 1432
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 962 838 1687 1431
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 880 774 1506 1287
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 419 365 713 607
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 620 524 1130 940
...FIRST_SWITCH_BLOCK/input[0]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 410 356 704 598
...FIRST_SWITCH_BLOCK/input[1]_to_...apuf_response_mutual_order_reg_reg[1]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[1]_to_...apuf_response_mutual_order_reg_reg[2]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[1]_to_...apuf_response_mutual_order_reg_reg[3]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[1]_to_...apuf_response_mutual_order_reg_reg[4]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[1]_to_...apuf_response_mutual_order_reg_reg[5]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[1]_to_...apuf_response_mutual_order_reg_reg[6]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[1]_to_...FSM_onehot_state_reg[3]/D 611 524 1123 944
...FIRST_SWITCH_BLOCK/input[1]_to_...FSM_onehot_state[0]_i_1/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[1]_to_...busy_INST_0/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 880 774 1506 1287
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 419 365 713 607
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 620 524 1130 940
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 410 356 704 598
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 689 594 1252 1054
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 770 665 1407 1185
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 867 761 1492 1273
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 963 839 1688 1432
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 962 838 1687 1431
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 716 610 1304 1089
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 712 606 1303 1087
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 961 822 1757 1471
...FIRST_SWITCH_BLOCK/input[1]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[2]_to_...apuf_response_mutual_order_reg_reg[1]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[2]_to_...apuf_response_mutual_order_reg_reg[2]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[2]_to_...apuf_response_mutual_order_reg_reg[3]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[2]_to_...apuf_response_mutual_order_reg_reg[4]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[2]_to_...apuf_response_mutual_order_reg_reg[5]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[2]_to_...apuf_response_mutual_order_reg_reg[6]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[2]_to_...FSM_onehot_state_reg[3]/D 611 524 1123 944
...FIRST_SWITCH_BLOCK/input[2]_to_...FSM_onehot_state[0]_i_1/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[2]_to_...busy_INST_0/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 880 774 1506 1287
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 419 365 713 607
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 620 524 1130 940
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 410 356 704 598
```

```
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 716 610 1304 1089
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 712 606 1303 1087
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 961 822 1757 1471
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 963 839 1688 1432
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 962 838 1687 1431
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 689 594 1252 1054
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 770 665 1407 1185
...FIRST_SWITCH_BLOCK/input[2]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 867 761 1492 1273
...FIRST_SWITCH_BLOCK/input[3]_to_...apuf_response_mutual_order_reg_reg[1]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[3]_to_...apuf_response_mutual_order_reg_reg[2]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[3]_to_...apuf_response_mutual_order_reg_reg[3]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[3]_to_...apuf_response_mutual_order_reg_reg[4]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[3]_to_...apuf_response_mutual_order_reg_reg[5]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[3]_to_...apuf_response_mutual_order_reg_reg[6]/CE 524 449 1016 850
...FIRST_SWITCH_BLOCK/input[3]_to_...FSM_onehot_state_reg[3]/D 611 524 1123 944
...FIRST_SWITCH_BLOCK/input[3]_to_...FSM_onehot_state[0]_i_1/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[3]_to_...busy_INST_0/I0 277 234 555 464
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 880 774 1506 1287
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 419 365 713 607
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 620 524 1130 940
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 410 356 704 598
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 716 610 1304 1089
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 712 606 1303 1087
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 961 822 1757 1471
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 689 594 1252 1054
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 1038 899 1832 1547
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 770 665 1407 1185
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 867 761 1492 1273
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 963 839 1688 1432
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 962 838 1687 1431
...FIRST_SWITCH_BLOCK/input[3]_to_...APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 961 837 1685 1429
# switch_block_delays apuf_driver/APUF_TEST_UNIT/FIRST_SWITCH_BLOCK/output
...FIRST_SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 335 281 645 535
...FIRST_SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 307 259 592 492
...FIRST_SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 412 343 788 649
...FIRST_SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 417 312 799 554
...FIRST_SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 264 227 500 422
...FIRST_SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 237 201 449 376
...FIRST_SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 344 284 680 557
...FIRST_SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 351 253 694 463
...FIRST_SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 408 340 785 646
...FIRST_SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 408 340 787 648
...FIRST_SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 306 258 600 498
...FIRST_SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 205 172 373 310
...FIRST_SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 438 361 863 705
...FIRST_SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 327 273 637 527
...FIRST_SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 259 215 547 449
...FIRST_SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[1].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 243 195 464 358
# for {set sb 1} {$sb <= 23} {incr sb} {
#     switch_block_delays apuf_driver/APUF_TEST_UNIT/SWITCH_BLOCKS[$sb].SWITCH_BLOCK/output
# }
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 362 303 692 572
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 362 303 692 572
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 363 304 694 573
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 153 136 268 231
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 152 135 268 231
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 151 134 265 229
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 258 220 494 413
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 259 221 498 416
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 367 308 702 580
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 354 300 669 558
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 195 169 360 305
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 195 169 359 304
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 290 246 526 439
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 291 247 530 441
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 463 386 872 716
...SWITCH_BLOCKS[1].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[2].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 464 387 876 719
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 281 242 508 428
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 254 216 457 382
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 430 359 800 659
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 425 354 797 656
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 349 288 676 552
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 496 420 973 807
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 432 361 833 685
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 412 354 790 661
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 364 303 688 565
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 364 303 690 567
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 266 226 495 412
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 162 143 270 231
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 287 241 544 450
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 259 219 491 407
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 289 243 546 452
...SWITCH_BLOCKS[2].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[3].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 286 240 542 448
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 270 226 534 442
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 175 149 339 284
```

```
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 273 229 549 453
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 277 204 557 381
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 440 363 854 698
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 439 362 851 696
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 444 366 858 701
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 236 194 437 358
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 119 102 233 197
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 364 309 676 564
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 278 240 499 422
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 280 231 503 395
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 413 344 818 673
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 240 202 479 397
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 242 203 490 406
...SWITCH_BLOCKS[3].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[4].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 350 255 686 459
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 382 319 728 597
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 421 353 812 669
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 460 381 865 707
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 467 350 879 613
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 225 190 466 390
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 318 266 667 551
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 387 325 782 648
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 391 300 790 576
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 160 140 273 232
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 158 138 269 229
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 158 138 269 229
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 270 206 519 363
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 377 313 707 581
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 370 308 698 573
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 366 305 689 566
...SWITCH_BLOCKS[4].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[5].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 371 274 700 471
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 505 423 930 767
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 433 361 811 666
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 407 337 745 593
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 334 274 602 472
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 242 208 417 352
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 282 242 486 410
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 579 449 1061 773
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 246 206 425 353
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 498 417 919 758
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 323 272 616 510
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 222 186 389 322
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 503 386 930 663
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 466 399 889 745
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 426 355 801 658
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 356 271 681 483
...SWITCH_BLOCKS[5].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[6].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 473 383 904 690
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 426 355 817 673
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 426 355 817 673
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 243 203 466 385
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 244 204 470 388
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 310 258 585 480
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 313 260 589 484
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 279 228 551 448
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 279 228 553 450
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 188 160 338 285
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 189 161 342 287
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 146 127 248 212
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 362 301 688 565
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 357 296 679 557
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 319 269 618 512
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 280 234 531 439
...SWITCH_BLOCKS[6].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[7].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 185 157 336 281
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 578 488 1085 901
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 260 223 474 399
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 504 424 937 777
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 292 254 505 430
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 232 204 408 350
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 333 284 634 530
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 437 367 830 687
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 437 367 832 689
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 442 372 835 692
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 437 367 832 689
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 332 283 630 527
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 266 229 492 415
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 282 245 509 432
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 362 307 687 573
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 363 308 689 575
...SWITCH_BLOCKS[7].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[8].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 362 307 687 573
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 200 173 366 310
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 199 172 363 308
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 240 204 455 381
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 240 204 454 380
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 237 201 458 385
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 302 254 599 497
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 410 341 804 662
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 411 342 808 665
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 346 287 693 569
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 345 286 693 569
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 240 202 488 405
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 241 203 492 408
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 481 402 935 772
```

```
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 481 402 935 772
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 318 264 632 521
...SWITCH_BLOCKS[8].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[9].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 341 294 654 550
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 347 288 677 557
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 242 204 480 398
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 348 289 679 558
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 252 208 525 432
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 136 119 251 214
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 347 288 682 560
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 242 204 477 396
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 243 205 481 399
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 240 202 479 397
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 345 286 676 556
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 133 116 248 212
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 136 119 252 215
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 238 202 446 375
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 239 203 450 377
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 411 342 792 652
...SWITCH_BLOCKS[9].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[10].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 412 343 796 655
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 360 298 687 563
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 148 128 255 216
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 443 371 844 696
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 439 367 843 694
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 367 305 690 567
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 367 305 692 569
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 269 228 497 414
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 165 145 272 233
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 196 167 352 295
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 196 167 351 294
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 293 246 548 454
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 290 243 544 450
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 282 235 537 443
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 332 281 636 528
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 441 369 837 690
...SWITCH_BLOCKS[10].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[11].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 197 168 352 295
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 405 343 785 655
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 319 275 599 506
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 456 386 907 755
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 280 207 559 383
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 431 360 824 678
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 280 234 531 438
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 252 213 500 416
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 151 127 273 228
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 457 378 860 704
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 415 347 796 656
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 520 434 1012 833
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 289 234 551 429
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 268 230 451 380
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 272 234 456 384
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 466 388 881 724
...SWITCH_BLOCKS[11].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[12].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 470 357 884 623
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 231 203 408 349
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 343 271 658 483
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 333 285 628 526
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 438 369 830 688
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 443 372 827 685
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 403 314 813 599
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 439 369 823 682
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 440 370 827 685
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 444 370 819 675
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 449 339 830 580
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 443 369 819 675
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 339 286 623 518
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 370 311 691 572
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 282 229 501 390
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 367 308 676 561
...SWITCH_BLOCKS[12].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[13].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 302 260 536 452
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 281 235 507 420
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 285 239 514 426
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 452 373 844 690
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 459 342 858 596
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 369 305 693 569
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 362 300 684 561
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 284 238 547 451
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 288 213 555 379
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 152 132 259 220
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 150 130 255 217
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 358 297 675 554
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 262 198 505 351
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 352 293 668 549
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 453 374 847 693
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 245 207 429 357
...SWITCH_BLOCKS[13].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[14].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 249 205 437 358
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 361 302 707 583
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 281 237 544 452
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 366 307 712 588
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 281 237 544 452
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 358 299 700 577
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 320 272 609 508
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 350 296 664 553
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 156 139 282 243
```

```
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 198 171 358 304
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 366 307 711 588
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 269 231 517 434
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 366 307 711 588
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 289 245 565 471
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 194 168 370 313
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 208 182 385 328
...SWITCH_BLOCKS[14].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[15].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 204 178 380 323
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 250 213 456 384
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 253 216 460 387
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 529 442 1031 852
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 256 218 457 383
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 333 279 710 589
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 438 363 912 751
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 463 386 950 783
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 257 222 528 445
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 400 331 789 648
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 296 248 593 491
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 291 245 609 506
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 389 322 804 661
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 323 269 645 533
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 322 268 643 531
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 338 282 635 525
...SWITCH_BLOCKS[15].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[16].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 335 279 631 521
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 496 416 915 755
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 388 327 727 603
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 282 244 487 411
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 426 355 804 661
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 281 241 491 413
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 335 286 598 501
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 422 356 777 645
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 255 217 464 387
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 366 313 694 581
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 260 228 459 392
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 467 394 879 729
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 235 204 420 356
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 469 395 885 733
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 391 333 748 623
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 293 253 539 455
...SWITCH_BLOCKS[16].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[17].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 449 381 839 700
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 197 171 362 307
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 77 61 140 113
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 369 310 704 582
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 370 311 708 585
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 460 384 861 710
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 385 278 738 494
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 248 214 433 366
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 251 217 437 369
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 263 225 501 419
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 298 211 575 366
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 369 310 700 579
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 340 285 652 540
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 370 311 718 594
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 341 271 668 494
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 265 227 513 430
...SWITCH_BLOCKS[17].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[18].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 266 228 517 433
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 484 407 922 765
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 402 340 772 644
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 426 356 809 668
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 389 321 732 584
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 135 118 249 213
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 236 198 475 393
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 249 208 485 401
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 148 122 258 213
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 434 363 831 684
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 358 297 679 557
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 483 404 906 748
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 488 373 917 653
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 285 239 537 444
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 198 170 350 294
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 287 243 575 477
...SWITCH_BLOCKS[18].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[19].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 291 218 583 405
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 478 399 919 758
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 227 188 487 401
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 424 316 828 576
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 416 346 813 669
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 307 260 581 487
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 174 147 358 299
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 268 195 560 382
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 311 264 588 493
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 220 190 378 320
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 282 236 550 453
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 260 196 508 353
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 495 414 935 772
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 487 415 931 778
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 219 189 403 340
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 432 325 834 582
...SWITCH_BLOCKS[19].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[20].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 487 415 932 779
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 404 342 791 659
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 478 406 923 769
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 261 225 478 403
```

```
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 482 381 931 697
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 491 423 916 772
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 593 502 1109 923
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 426 358 832 689
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 598 471 1120 828
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 574 482 1077 891
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 342 292 607 510
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 490 411 923 764
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 344 283 611 483
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 502 420 922 762
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 296 255 504 426
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 215 188 392 334
...SWITCH_BLOCKS[20].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[21].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 300 253 512 427
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 426 364 825 689
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 337 292 632 535
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 504 426 962 799
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 511 395 976 705
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 192 166 382 323
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 190 164 378 320
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 466 389 940 775
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 194 162 386 321
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 316 267 652 541
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 315 266 648 538
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 265 221 553 455
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 344 249 692 463
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 473 393 939 775
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 466 388 930 767
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 298 253 599 502
...SWITCH_BLOCKS[21].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[22].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 392 301 801 585
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I1 193 167 358 303
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I1 193 167 357 302
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I1 207 181 372 317
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[0]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I1 287 243 550 458
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I0 365 306 707 584
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I0 155 138 281 242
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I0 160 143 285 247
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[1]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I0 164 147 290 251
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I5 221 194 421 359
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I5 365 306 710 587
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I5 291 247 567 473
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[2]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I5 365 306 710 587
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[0].EQUAL_PATH_MUX/Q_INST_0/I2 281 237 557 463
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[1].EQUAL_PATH_MUX/Q_INST_0/I2 316 268 601 501
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[2].EQUAL_PATH_MUX/Q_INST_0/I2 322 274 604 505
...SWITCH_BLOCKS[22].SWITCH_BLOCK/output[3]_to_...SWITCH_BLOCKS[23].SWITCH_BLOCK/EQUAL_PATHS[3].EQUAL_PATH_MUX/Q_INST_0/I2 197 171 373 316
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[0]_to_...APUF_ARBITER/FF_2_1/Q_reg/D 301 251 620 510
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[0]_to_...APUF_ARBITER/FF_3_1/Q_reg/D 363 303 728 600
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[0]_to_...APUF_ARBITER/FF_4_1/Q_reg/D 310 258 605 497
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[1]_to_...APUF_ARBITER/FF_2_1/Q_reg/C 323 275 600 503
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[1]_to_...APUF_ARBITER/FF_3_2/Q_reg/D 413 349 791 660
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[1]_to_...APUF_ARBITER/FF_4_2/Q_reg/D 386 324 776 645
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[2]_to_...APUF_ARBITER/FF_3_1/Q_reg/C 319 271 633 528
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[2]_to_...APUF_ARBITER/FF_3_2/Q_reg/C 319 271 633 528
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[2]_to_...APUF_ARBITER/FF_4_3/Q_reg/D 284 239 547 454
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[3]_to_...APUF_ARBITER/FF_4_1/Q_reg/C 254 214 471 392
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[3]_to_...APUF_ARBITER/FF_4_2/Q_reg/C 254 214 471 392
...SWITCH_BLOCKS[23].SWITCH_BLOCK/output[3]_to_...APUF_ARBITER/FF_4_3/Q_reg/C 254 214 471 392
```

References

[1] *Artix-7*. Apr. 1, 2020. URL: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html.

[2] S. V. Sandeep Avvaru, Ziqing Zeng, and Keshab K Parhi. "Homogeneous and Heterogeneous Feed-Forward XOR Physical Unclonable Functions". eng. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 2485–2498. ISSN: 1556-6013.

[3] Christoph Böhm and Maximilian Hofer. *Physical Unclonable Functions in Theory and Practice*. Springer-Verlag New York, 2013.

[4] *ChipWhisperer Software*. Apr. 1, 2020. URL: https://github.com/newaetech/chipwhisperer.

[5] *ChipWhisperer Wiki*. Apr. 1, 2020. URL: https://wiki.newae.com/Main%5C_Page.

[6] *CW1173 ChipWhisperer-Lite*. Apr. 1, 2020. URL: https://wiki.newae.com/CW1173_ChipWhisperer-Lite.

[7] *CW305 Artix FPGA Target*. Apr. 1, 2020. URL: https://wiki.newae.com/CW305_Artix_FPGA_Target.

[8] E. Dubrova. "A Reconfigurable Arbiter PUF with 4 x 4 Switch Blocks". In: *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*. May 2018, pp. 31–37. DOI: 10.1109/ISMVL.2018.00014.

[9] E. Dubrova et al. "CRC-PUF: A Machine Learning Attack Resistant Lightweight PUF Construction". In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. June 2019, pp. 264–271. DOI: 10.1109/EuroSPW.2019.00036.

[10] Elena Dubrova, Martin Brisfors, and Bernhard Degen. "A Model of Arbiter PUF with 4 x 4 Switch Blocks".

[11] Ilze Eichhorn, Patrick Koeberl, and Vincent van der Leest. "Logically reconfigurable PUFs: memory-based secure key storage". In: *Proc. of the sixth ACM workshop on Scalable Trusted Computing*. STC '11. Chicago, Illinois, USA: ACM, 2011, pp. 59–64. ISBN: 978-1-4503-1001-7. DOI: 10.1145/2046582.2046594. URL: http://doi.acm.org/10.1145/2046582.2046594.

[12] Yansong Gao and Damith C. Ranasinghe. *R^3PUF: A Highly Reliable MemRistive Device based Reconfigurable PUF*. Tech. rep. ArXive Technical report, Feb. 2017.

[13] Wei Ge et al. "FPGA implementation of a challenge pre-processing structure arbiter PUF designed for machine learning attack resistance". eng. In: *IEICE Electronics Express* 17.2 (2020). ISSN: 13492543. URL: http://search.proquest.com/docview/2349877475/.

[14] Rekha Govindaraj, Swaroop Ghosh, and Srinivas Katkoori. “Design, Analysis and Application of Embedded Resistive RAM based Strong Arbiter PUF”. eng. In: *IEEE Transactions on Dependable and Secure Computing* (2018), pp. 1–1. ISSN: 1545-5971.

[15] K. Kursawe et al. “Reconfigurable Physical Unclonable Functions - Enabling technology for tamper-resistant storage”. In: *Proc. of IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'09)*. July 2009, pp. 22–29. DOI: `10.1109/HST.2009.5225058`.

[16] Daihyun Lim. “Extracting secret keys from integrated circuits”. MA thesis. Massachusetts Institute of Technology, 2004.

[17] M. Majzoobi, F. Koushanfar, and M. Potkonjak. “Testing Techniques for Hardware Security”. In: *2008 IEEE International Test Conference(ITC)*. Vol. 00. Oct. 2009, pp. 1–10.

[18] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. “Lightweight secure PUFs”. eng. In: *Proceedings of the 2008 IEEE/ACM International Conference on computer-aided design*. ICCAD '08. IEEE Press, 2008, pp. 670–673. ISBN: 9781424428205.

[19] Phuong Ha Nguyen et al. *The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks*. Cryptology ePrint Archive, Report 2018/350. 2018.

[20] Ioannis Papakonstantinou and Nicolas Sklavos. “Physical Unclonable Functions (PUFs) Design Technologies: Advantages and Trade Offs”. In: Jan. 2018, pp. 427–442. ISBN: 978-3-319-58423-2. DOI: `10.1007/978-3-319-58424-9_24`.

[21] Durga Prasad Sahoo et al. “A Multiplexer-Based Arbiter PUF Composition with Enhanced Reliability and Security”. eng. In: *IEEE Transactions on Computers* 67.3 (2018), pp. 403–417. ISSN: 0018-9340.

[22] Soubhagya Sutar, Arnab Raha, and Vijay Raghunathan. “D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication in Embedded Systems”. In: *Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. CASES '16. Pittsburgh, Pennsylvania: ACM, 2016, 12:1–12:10. ISBN: 978-1-4503-4482-1. DOI: `10.1145/2968455.2968519`. URL: `http://doi.acm.org/10.1145/2968455.2968519`.

[23] *Vivado*. Apr. 1, 2020. URL: `https://www.xilinx.com/products/design-tools/vivado.html`.

[24] Nils Wisiol and Marian Margraf. “Why attackers lose: design and security analysis of arbitrarily large XOR arbiter PUFs”. eng. In: *Journal of Cryptographic Engineering* 9.3 (2019), pp. 221–230. ISSN: 2190-8508.