EasyChair Preprint

№ 8661

Reduction of Bitstream Size for Low-Cost iCE40 FPGAs

Clemens Fritzsch, Joern Hoffmann and Martin Bogdan

August 11, 2022

Reduction of Bitstream Size for Low-Cost iCE40 FPGAs

Clemens Fritzsch, Jörn Hoffmann and Martin Bogdan
Neuromorphic Information Processing, Leipzig University, Leipzig, Germany
Email: {fritzsch, jhoffmann, bogdan}@informatik.uni-leipzig.de

***Abstract*—Reducing the bitstream size is important to lower external storage requirements and to speed-up the reconfiguration of field-programmable gate arrays (FPGAs). The most common methods for bitstream size reduction are based on dedicated hardware elements or dynamic partial reconfiguration. All of these properties are usually missing in low-cost FPGAs such as the Lattice iCE40 device family.**

In this paper we propose a lightweight compaction approach for iCE40 FPGAs. We present five methods for bitstream compaction: two adapted and three new. The methods work directly on the bitstream by removing unnecessary data and redundant commands. They are applicable independent of the synthesis toolchain and require neither repetition of synthesis steps nor modifications of the target system. Although our focus is on iCE40 devices, we additionally discuss the conditions for applying our approach to other targets.

All five methods were implemented in an open-source compaction tool. We evaluate our approach with an iCE40 HX8K FPGA by synthesizing and compacting various projects. As a result, we achieve a reduction in bitstream size and reconfiguration time by up to 79 %.

I. Introduction

The bitstream of a field-programmable gate array (FPGA) contains the configuration data of the internal fabric. Its size is a crucial factor for the cost and the performance of a system.

The amount of necessary external storage and with it the cost for additional hardware components are influenced by the bitstream size. The importance is especially high for applications that employ multiple configurations, like the system with dynamically self-selected coprocessors presented by Gonzalez et al. [1]. Numerous large bitstreams can necessitate a larger flash memory chip and increase the cost of the system.

Furthermore, the bitstream size determines the amount of data transferred and the time needed to reconfigure the FPGA. This is also crucial in the research field of Evolvable Hardware (EHW) [2], where 250 000 reconfigurations can be required for a single experiment [3]. Consequently, short reconfiguration times are paramount, even if simple low-cost FPGAs such as the Lattice iCE40 are used [4].

Most common approaches for bitstream size reduction apply compression algorithms [5]–[7]. Such approaches can be classified by whether the bitstream is decompressed on-chip or off-chip.

On-chip approaches require additional properties of the FPGA. Yan et al. [5] evaluated several device-independent compression algorithms. Their on-chip variants, however, require dynamic partial reconfiguration. Many Xilinx FPGAs (e.g. Spartan 3 and the 7 Series) reduce their bitstream size by applying the same configuration data to multiple parts of the FPGA [8]. This, however, requires dedicated hardware in form of the multiple frame write register. Intel Stratix 10 includes the decompression directly in the hardware that manages the device configuration [9].

Dedicated decompression hardware and dynamic partial reconfiguration are common for high-end and mid-end FPGAs, but are usually missing in low-cost FPGAs.

Off-chip approaches do not depend on the properties of the target FPGA. As a result they are also feasible for low-cost FPGAs. Unfortunately, off-chip approaches cannot reduce the reconfiguration time since the uncompressed bitstream still has to be transmitted to the FPGA. A simple off-chip approach is the application of general-purpose data compression programs like gzip [10]. Koch et al. [6] as well as Yan et al. [5] have shown that it reduces the bitstream size more than on-chip methods in nearly all cases. Wolf and Lasser, on the other hand, encode the length between set bits to provide a lightweight bitstream compression algorithm for Lattice iCE40 FPGAs [7].

iCE40 devices are low-cost and low-power FPGAs produced by Lattice. Shah et al. presented a completely open-source synthesis toolchain for all iCE40 devices [11]. This enables a level of transparency and reproducibility that make iCE40 FPGAs highly suited for scientific research [4], [12], [13] and security sensitive applications [14].

This paper introduces methods to reduce the bitstream size and the reconfiguration time for low-cost Lattice iCE40 FPGAs. They require neither the repetition of synthesis steps nor special properties of the target FPGA or its environment.

Our main contributions are:

- Two adapted and three new methods to reduce the bitstream size of Lattice iCE40 FPGAs. The methods are independent of the synthesis toolchain and do not require any modification of the target system.
- Implementation of the five methods for iCE40 HX8K, HX1K, and LP1K devices in an open-source tool, that is available at https://github.com/nmi-leipzig/compact_bitstream.
- Evaluation of the size reduction capability of the five compaction methods with five example projects in combination with the vendor toolchain and the open-source toolchain. We also measured the effect of the size reduction on the reconfiguration time.

II. Background iCE40

Lattice iCE40 FPGAs provide between 384 and 7680 lookup tables (LUTs) for logic design. In addition, they offer up to 16 KiB of block RAM (BRAM) that can be used for memory cells. Most devices also contain one or two phase-locked loops. Only the Ultra Plus subfamily offers more hard blocks, like digital signal processors, internal oscillators, or an I²C block.

Like most modern FPGAs, iCE40 devices are based on SRAM cells. The SRAM, in turn, is split between configuration memory (CRAM) and block RAM. BRAM represents only the contents of the embedded memory, while CRAM represents every other aspect of the configuration. CRAM and BRAM are each split in four banks. Each of which correspond to a quadrant of the iCE40 physical chip.

During the configuration process, the chip clears the whole CRAM while the BRAM is left unchanged [15]. Afterwards the bitstream is loaded into the FPGA.

The bitstream for iCE40 FPGAs consists of two sections: a comment section and a series of commands [15]. Only the commands affect the configuration.

To write CRAM data, a couple of selection commands followed by the actual write command are necessary. The selection commands set the bank number, width, offset, and height. Bank number, width, and offset select the actual chip resources while the height defines the number of rows to be written. The subsequent write command starts the writing process and is followed by the actual data. Writing block RAM data follows the same sequence of commands but replaces the final command with a BRAM write command.

III. Compaction Methods

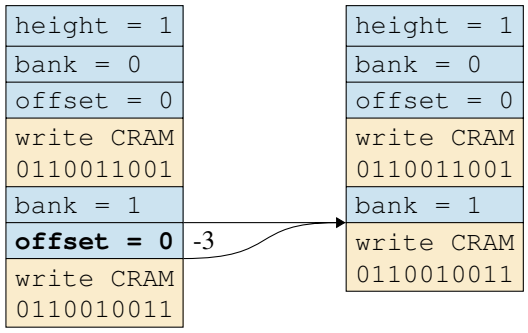
Our approach to reduce the bitstream size consists of five methods. Two of them are options that are built into the proprietary vendor toolchain. We have reimplemented them so that they can be used independently of the toolchain. The remaining three methods are particular to our approach. In short, they take advantage of the flexible bitstream format of iCE40 FPGAs and exploit the fact that the CRAM is cleared during the configuration process.

All five methods belong in the category of compaction rather than compression since no explicit reversal of the size reduction is required. The resulting bitstreams can be used directly, do not require any additional processing steps, and pose no further requirements on the target system.

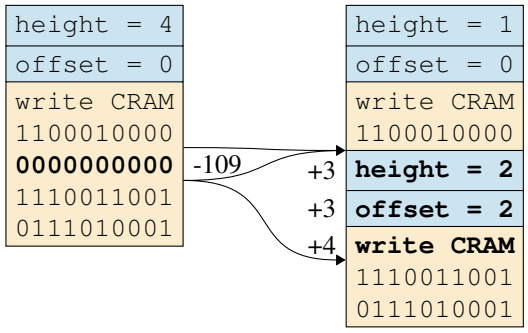
A. Adapted Built-in Options

The synthesis toolchain provided by Lattice is called iCEcube2. It provides two applicable options: one to skip the comment section and one to skip selected BRAM banks. The two options are not strictly intended for the purpose of size reduction, but can lead to savings, as shown in Section V.

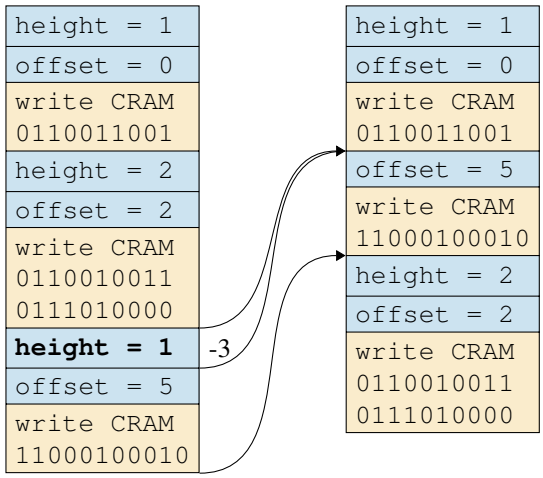
The comment section at the beginning has no influence on the configuration and can safely be left out in order to reduce the bitstream size.



(a) Value persistence: Omit redundant selection commands.



(b) Zero row skipping: Exclude zero rows by splitting write commands. The bank width of 109 bytes is specific for iCE40 HX8K FPGAs.



(c) Chunk sorting: Group writes of the same size and apply value persistence (see a).

Fig. 1. Examples for the newly proposed compaction methods. The left side shows the selection commands and write commands before applying the respective method. The right side shows the result. The deleted respectively inserted parts are bold. Their sizes in bytes are indicated next to them.

The second option to skip BRAM banks is more interesting. It is used for cases where either the initial content is not relevant or the content is to be reused. The latter is the case, if a configuration is subsequently replaced by another. Since the BRAM banks are not cleared during reconfiguration, data can be transferred from one configuration to the next. iCEcube2 facilitates this by skipping the BRAM write commands for selected banks. Although it is possible to determine from the

bitstream alone whether a BRAM bank is being accessed, it cannot be determined without further information whether the access is based on the initial or transferred BRAM content. Therefore, BRAM banks still have to be selected manually.

B. Utilization of Value Persistence

Our first novel method takes advantage of the fact that previously set selection values are kept. The values set for bank number, width, height, and offset are valid until a command explicitly sets a new value. Hence, it is only necessary to set a new value if the next write command requires a value that differs from the current one (Fig.1a). This persistence of values is already used in bistreams created by iCEcube2, although it is not applied in every possible situation. We therefore use this property more consistently in our approach to skip unnecessary commands and to reduce the bitstream size.

C. Zero Row Skipping

CRAM can be written in chunks by manipulating height and offset values. Due to the clearing of CRAM during configuration, a not explicitly written CRAM row will have all values set to zero. Consequently, all zero rows can be skipped without changing the configuration (Fig.1b). Zero rows are skipped by not including them in any chunk.

The skipping of rows introduces a small overhead for the additional selection and write commands. Fortunately, the overhead[1] is always less than the data in one row. As a result, the bitstream size is always reduced by skipping zero rows.

Although BRAM can also be written in chunks, there is no practical advantage over skipping whole BRAM banks (see Section III-A). Memory blocks within the BRAM are written in columns. Therefore, it is necessary to write every single row, i. e. the whole bank, in order to store the values of a single block. While it is possible that only a part of a memory block is used, the information about that has to be provided manually. The reason is the same as for the manual selection in BRAM bank skipping. The manual provision of memory regions is labor-intensive, error-prone, and not practical.

Attempts to write only the first part of BRAM rows by manipulating the bank width value resulted in unusable BRAM content. Consequently, sub row granularity was not further investigated, neither for BRAM nor CRAM.

D. Chunk Sorting

CRAM chunks can be written in an arbitrary order regarding their offset. This allows to write the chunks of a single CRAM bank ordered by their height. Due to the persistence of the selection values, set height commands with the same value can be skipped (Fig.1c). Consequently, only one set height command per unique height value is required instead of one per chunk. That way some overhead caused by zero row skipping is mitigated.

[1]The overhead is 10 bytes or 80 bits for the additional CRAM write command and setting new height and offset values, compared to the minimal CRAM bank width of 182 bits for iCE40 LP384.

E. Implementation

All five compaction methods were implemented in a Python application. This compaction tool takes an existing bitstream file, applies the compaction methods, and writes the resulting bitstream to an output file. As the tool works directly on an existing bitstream file, it is independent from the used synthesis toolchain and does not require the repetition of any synthesis steps.

Currently, bitstreams for iCE40 HX8K, HX1K, and LP1K FPGAs can be compacted by the tool. The selection stems from the availability of suitable test device to the authors at the time of the tool creation. Even so, the Python code was written with the aim of extensibility to simplify the addition of further iCE40 devices.

F. Applicability

The implementation of all five compaction methods relies on the iCE40 bitstream format. All devices from the iCE40 family share the bitstream format, therefore the five compaction methods can fundamentally be applied to them. The implementation of each method has to still be checked for each device. iCE40 LP384 for example has a CRAM bank width of 182 bits. Hence, each chunk has to contain a multiple of four rows. Furthermore, iCE40 LP384 has no BRAM. Therefore, BRAM skipping will not reduce the bitstream size.

The successful application of the five compaction methods to other FPGA device families heavily depends on their bitstream format. The bitstream format needs to meet the following conditions:

- Comment skipping
  - Bitstream contains comment
  - Comment can be excluded from bitstream
  - No influence of the comment on the configuration
- BRAM skipping
  - Bitstream contains BRAM data
  - BRAM data can be excluded from bitstream
- Utilization of value persistence
  - Usage of selection commands
  - Persistence of selection values
  - No enforcement of explicit selection commands before a write command
- Zero row skipping
  - Clearing of CRAM during configuration
  - CRAM can be written partially
- Chunk sorting
  - Utilization of value persistence possible
  - Order of write commands can be chosen

A promising candidate for adaption is the Anlogic Eagle device family. Its bitstream format is reasonably well documented in Project Tang [16]. The bitstream contains a comment and the CRAM is cleared during configuration [17, p. 59].

IV. Experimental Setup

To evaluate the proposed methods, five example projects were chosen:

- *blinky*: A simple example design that is included in the open-source toolchain.
- *ehw*: An evolved design to show the applicability to Evolvable Hardware.
- *attosoc*: A minimal RISC-V system on a chip, that is used for tests in the open-source toolchain.
- *updater*: A design that receives a new configuration, decrypts it (AES) and writes it to flash. This represents a typical secondary configuration.
- *picosoc*: A complete RISC-V system on a chip [18] with high resource utilization to find the limitations of our methods.

All projects were synthesized by the vendor toolchain iCEcube2 and the open-source toolchain presented by Shah et al. [11].

For each *original* bitstream, two compacted variants were created with our compaction tool. The *built-in* variant only employed the two built-in options, while the *compact* variant used all five proposed compaction methods. This serves to differentiate between the effect of the existing built-in options and the three newly proposed methods. The generation of the compacted bitstreams always took less than 0.3 s on an Intel i5-7200U CPU.

For time measurements, an iCE40 HX8K breakout board (ICE40HX8K-B-EVN) was used. The breakout board is equipped with an FTDI USB interface which was connected to a PC system. The configuration process and its measurement were executed directly by the compaction tool. Each configuration process for each of the 30 bitstreams was repeated 10 000 times to compensate for interference by the operating system or the like. The opening and closing of the USB connection was not included in the time measurement. Practical applications commonly connect to the FPGA directly so the exclusion of the USB overhead produces more applicable results.

Finally, the five *original* bitstreams were compressed with the icecompr tool by Wolf and Lasser [7] and with gzip[2] [10]. The off-chip compression tool icecompr is the only other available size reduction method for iCE40 bitstreams, while gzip had shown great size reduction capabilities for bitstreams in previous work [5], [6].

V. Experimental Results

The bitstream size was reduced for every example project and with both toolchains. Table I shows the sizes for all 30 bitstreams and the used lookup table and BRAM resources.

The built-in options as well as all five methods together lead to a reduction of the bitstream size. The built-in options generate distinct bitstream sizes depending on the number of skipped BRAM banks. This leads to a lower limit of 119 kB.

The size of the *compact* bitstream depends heavily on the project and the toolchain. As shown in Table I, *ehw* was reduced substantially by 79.7 % and 72.8 % for iCEcube2 and the open-source toolchain respectively. In contrast, only a small, albeit reasonable reduction was achieved for *picosoc*. This is caused by the high resource utilization of this project.

Beside LUT and BRAM utilization, the choice of routing resources also influences the bitstream size. The open-source toolchain uses global distribution network for *blinky* while iCEcube2 uses only local connections. Since the global distribution network requires further and more distributed configuration bits, the *compact* size more than doubles.

Fig.2 shows the linear relation between reconfiguration time and bitstream size. There is only a very small fraction of the reconfiguration time that is independent from the bitstream size. Consequently, statements for the bitstream sizes are also applicable to the reconfiguration times.

Although the Python interpreter and the USB connection had a large potential for interference, the measured reconfiguration times for each bitstream deviate less than 1.4 ms from the mean.

[2]Version 1.9, compression level `--best`

TABLE I
Bitstream Sizes as Synthesized by iCEcube2 (IC) and the open-source toolchain (OS) in Comparison to the Compacted Variants, Supplemented by the Used FPGA Resources.

Project	Tool-chain	Size (bytes) original	built-in	compact	LUT (7680)	BRAM (32)
blinky	IC	135180	118642	11071	35	0
	OS	135100	118642	23489	38	0
ehw	IC	135180	118642	27402	462	0
	OS	135100	118642	36765	463	0
attosoc	IC	135180	126872	93865	1776	4
	OS	135100	122760	57149	1686	4
updater	IC	135180	118642	94674	2474	1
	OS	135100	122760	74322	2579	2
picosoc	IC	135180	130984	128947	5219	6
	OS	135100	126872	126293	5153	6

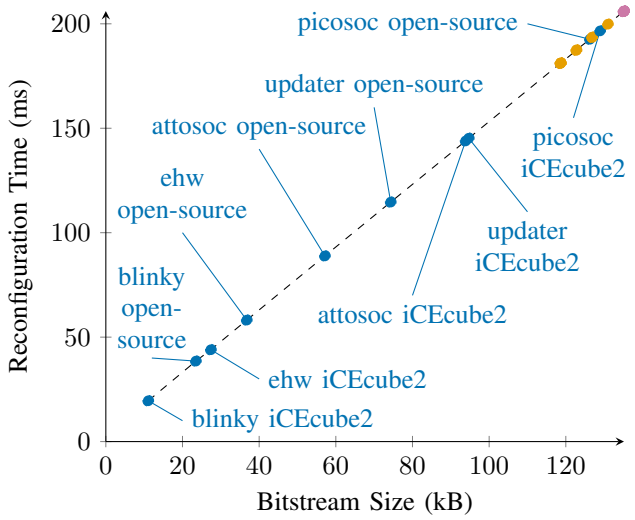


Fig. 2. Reconfiguration time as a function of bitstream size. The time value is the arithmetic mean over all measured reconfiguration times of bitstreams of that size. The colors encode the compact, built-in, and original variants. The compact variants are labeled with project and toolchain.

TABLE II
SIZES OF COMPACTED AND COMPRESSED BITSTREAMS RELATIVE TO THE ORIGINAL SIZE AS SYNTHESIZED BY ICECUBE2 (IC) AND THE OPEN-SOURCE TOOLCHAIN (OS), SUPPLEMENTED BY THE FPGA RESOURCES UTILIZATION.

Project	Tool-chain	Ratio to original size (%)			LUT (%)	BRAM (%)
		compact	icecompr	gzip		
blinky	IC	8.2	0.6	0.6	0.5	0
	OS	17.4	1.7	0.6	0.5	0
ehw	IC	20.3	4.0	3.0	6.0	0.0
	OS	27.2	5.1	3.6	6.0	0.0
attosoc	IC	69.4	21.2	20.3	23.1	12.5
	OS	55.0	16.6	15.3	22.0	12.5
updater	IC	70.0	26.3	24.9	32.2	3.1
	OS	55.0	23.7	23.1	33.6	6.3
picosoc	IC	95.4	49.5	46.7	68.0	18.8
	OS	93.5	46.3	43.7	67.1	18.8

The size relative to the original size is shown in Table II for the *compact* bitstreams and the compressed versions. Both compression algorithms achieve better size reductions than the compaction methods. They can reduce *picosoc* by more than 50 % and *blinky* even by 99.4 %. While gzip is always better than icecompr, the difference is small in relation to the overall size reduction.

VI. DISCUSSION

Our results show that both the bitstream size and reconfiguration time were reduced for all projects, in most cases significantly. As expected, bitstream size and reconfiguration time are almost directly proportional. Therefore, our method to reduce the bitstream size is a very effective approach for decreasing the reconfiguration time.

When comparing the *compact* bitstream size and the resource utilization in Table I, a direct, but not linear relation becomes evident. The missing aspect is the distribution of used resources between the respective CRAM and BRAM banks. *Updater* synthesized with iCEcube2 uses less resources, nevertheless its *compact* size is larger than *updater* synthesized with the open-source toolchain. The open-source toolchain spreads the resources over less CRAM and BRAM banks, thus the proposed compaction methods are more effective.

Especially, applications that use multiple configurations profit from our compaction methods. While the most resource intensive configurations of the application can be compacted only slightly, smaller, secondary configurations benefit strongly. *Original picosoc* and *updater* cannot fit in a 256 kB flash memory together. The *compact* versions, however, have even enough memory left to add *ehw*.

In addition, applications that require numerous reconfigurations can benefit. The duration of Evolvable Hardware experiments with 250 000 reconfigurations (e.g. [3]) is shortened by more than 11 h.

The off-chip compression by icecompr and gzip achieve better size reduction. Nevertheless, they increase the reconfiguration time and cost for additional hardware, because the bitstream has to be decompressed externally. The lightweight icecompr reduces the bitstream sizes nearly as well as the more resource intensive gzip. This suggests, that lightweight hardware decompression support would result in the best compromise between cost and size reduction. Additionally, the compressed sizes increase if the *compact* sizes increase. This indicates that the harder to compact bitstreams indeed contain more information and are not just unfortunate cases.

As Yan et al. [5] point out, better size reduction at the cost of low throughput is undesirable for productive systems. Our compaction methods do never impede on the throughput of the configuration data since they do not require a decompression step. In the absence of hardware decompression, they are the best solution for bitstream size reduction and speed-up of reconfiguration times.

VII. CONCLUSION

In this work we proposed two adapted and three new methods to compact the configuration bitstream of iCE40 FPGAs which lack the requirements for sophisticated compression methods. We have implemented these methods for iCE40 HX8K, HX1K, and LP1K devices. The methods and our tool can be used independently from the synthesis toolchain.

The results show that the bitstream size is reduced substantially in most cases. This also significantly reduces the time needed to reconfigure the device.

Our methods requires neither repetition of synthesis steps nor modification of the target device. The compaction methods can easily be integrated in existing productive workflows.

Future work will characterize the relationship between resource utilization, bitstream entropy, and compaction results. This can be combined with modifications of the synthesis toolchain to improve the effect of compaction. One possibility is the inclusion of resource distribution metrics into the place and route algorithms.

Furthermore, the applicability to other low-cost FPGAs, like the Anlogic Eagle device family, will be evaluated.

DATA AVAILABILITY

The time measurements and the bitstreams that support Table I, Table II, and Fig.2 are openly available at https://zenodo.org/record/6735068.

ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research (BMBF, 01IS18026B) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".

REFERENCES

[1] I. Gonzalez, E. Aguayo, and S. Lopez-Buedo, "Self-reconfigurable embedded systems on low-cost fpgas," *IEEE Micro*, vol. 27, no. 4, pp. 49–57, 2007.

[2] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 183–215, 2011.

[3] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics," in *Evolvable Systems: From Biology to Hardware*. Springer Berlin Heidelberg, 1997, pp. 390–405.

[4] D. Whitley, J. Yoder, and N. Carpenter, "Resurrecting FPGA intrinsic analog evolvable hardware," in *The 2021 Conference on Artificial Life*. MIT Press, 2021.
[5] J. Yan, J. Yuan, P. H. W. Leong, W. Luk, and L. Wang, "Lossless compression decoders for bitstreams and software binaries based on high-level synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2842–2855, oct 2017.
[6] D. Koch, C. Beckhoff, and J. Teich, "Bitstream decompression for high speed FPGA configuration from slow memories," in *2007 International Conference on Field-Programmable Technology*. IEEE, dec 2007.
[7] C. Wolf and M. Lasser, "Project IceStorm," http://bygone.clairexen.net/icestorm/.
[8] Xilinx Technical Staff, *7 Series FPGAs Configuration User Guide (UG470)*, Xilinx, 2018.
[9] I. T. Staff, *Intel Stratix 10 Configuration User Guide*, 2022.
[10] GNU Project. (2018) Gnu gzip 1.9. [Online]. Available: https://git.savannah.gnu.org/cgit/gzip.git/tag/?h=v1.9
[11] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+nextpnr: An open source framework from verilog to bitstream for commercial FPGAs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, apr 2019.
[12] Q. A. Ahmed, "Hardware trojans in reconfigurable computing," in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, oct 2021.
[13] C. Fibich, M. Horauer, and R. Obermaisser, "Device- and temperature dependency of systematic fault injection results in artix-7 and iCE40 FPGAs," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, feb 2021.
[14] A. Huang, "Betrusted: Improving security through physical partitioning," *IEEE Pervasive Computing*, vol. 19, no. 2, pp. 13–20, apr 2020.
[15] Lattice Semiconductor Technical Staff, *iCE40 Programming and Configuration*, Lattice Semiconductor, Mar. 2020.
[16] M. Milanović and I. Zheng, "Project tang," https://github.com/mmicko/prjtang.
[17] Anlogic Technical Staff, *Anlogic Technology EAGLE Series FPGA Data Sheet (DS300)*, Anlogic, 2018.
[18] C. Wolf. (2021) Picorv32. [Online]. Available: https://github.com/cliffordwolf/picorv32