qMC: Formal Model Checking Verification Framework For Superconducting Logic

Mustafa Munir*, Kunal Sheth[†], Aswin Gopikanna[‡], Arash Fayyazi[§] and Shahin Nazarian[¶]
Ming Hsieh Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, CA
(*mmunir, [†]kunalche, [‡]gopikann, [§]fayyazi, [¶]shahin.nazarian)@usc.edu

I. Problem Description

Our project's main goal is to develop a model checking based formal verification tool for Single Flux Quantum (SFQ) technology. We modified the BMC (Bounded Model Checking) algorithm to make it compatible with our SFQ models. This will aid professor Nazarian's research group in the development of their qMC tool.

II. Project Timeline

06/01/2020 - 06/07/2020:

Reading papers from the field of Single Flux Quantum technology and Model Checking, installing the Yosys tool, and running some examples in CMOS technology.

Went through all of the Yosys material from last year and familiarized ourselves with the Boundary Model Checking (BMC) tool.

Familiarized ourselves with the prior work in the field of formal model checking verification and superconducting flux quantum technologies.

Began working on translating the blif file as we have got Yosys to work, but receive an error when running the sby config command.

06/08/2020 - 06/19/2020:

Designed SFQ logic gates for AND, OR, NOR, XOR, Inverter, DFF, and Splitter in Verilog.

Achieved the translation of the blif file (e.g., generated by synthesis tools) to SystemVerilog files, which are accepted by Yosys. We used the formal assertions on the SFQ logic gates.

06/20/2020 - 06/26/2020:

Formally verified (model checking) the generated SystemVerilog file using SymbiYosys BMC and displayed the output.

06/27/2020 - 07/10/2020:

Automize testbench generation, SystemVerilog file generation, and results generation.

Investigate and come up with the best verification strategy.

Write formal assertions to verify Kogge-Stone Adder, Array Multiplier, and Integer Divider.

07/10/2020 - 07/14/2020

Upload our work including the code to Gitlab and write a README file to briefly explain our work done in the semester.

Also submit our final report and present our project to Professor Nazarian and our Ph.D. student advisor Arash Fayyazi.

***Abstract*—This paper proposes a verification framework called quantum Model Checker (qMC) for single-flux quantum (SFQ) circuits using formal techniques. SFQ circuits have the potential to replace the CMOS circuits as they possess a theoretical potential of six orders of magnitude reduction in power accompanied with one order of magnitude of higher speed. Despite its benifits, the SFQ lacks a model checking based formal verification tool. qMC is an automated formal verification tool that constructs a systemverilog testbench consisting of formal assertions to verify the SFQ-specific properties of the circuits and produce system correctness results and counterexamples using Model Checking (MC). The proposed framework is implemented for various SFQ combinational circuits: Kogge-Stone Adders (KSA), integer dividers, array multipliers, and counter.**

***Keywords*—Single-Flux Quantum (SFQ) , Model Checking, Formal Verification, System Verification, Verilog, SystemVerilog.**

III. Introduction

The end of Moore's Law has lead to a desire to find alternative technologies that can replace the widespread use of CMOS transistors. Currently CMOS technologies have a widespread set of tools available in industry, while other technologies do not have the same support. This causes other technologies to not be viable to use in a widespread sense even if they have benefits over CMOS technology. Despite the challenges other technologies face there are still some promising technologies like Superconducting Electronics, carbon nanotubes, and magnetically sensitive (spin) transistors.

Carbon nanotube technology holds promise, but it faces many hurdles like scalability and fabrication challenges. Magnetically sensitive (spin) transistors are another option. Spin transistors provide better energy efficiency than CMOS transistors for information processing, but they also are less reliable and have difficulty in the reading and writing of information.

Whatever new technology competes with CMOS must have the necessary tools, be low power, and be high speed. A type of superconducting technology called Single Flux Quantum technology meets this need for low power consumption and extremely high operating speed, but does not have the necessary tool support. SFQ technology operates at cryogenic temperatures and has a reduction in power consumption of six orders of magnitude when compared to even high-end CMOS circuits, while still providing an improvement in speed. The reason SFQ technology can provide high operating speeds and low power consumption is because of Josephson junctions. Josephson junctions are a quantum mechanical device that is able to operate at high frequencies and has low switching energy thereby reducing power consumption.Josephson junctions are created through using two superconducting electrodes that are separated by an insulating barrier.

SFQ circuits still do not have an optimal design approach, unlike CMOS circuits. This makes SFQ difficult to use for large-scale circuits. Despite this shortcoming SFQ technology's unique properties allow for it to be a possible alternative to CMOS when the technology matures. As SFQ technology matures the development of EDA tools for and robust verification tools for it become more necessary. SFQ technology features different variations including Rapid Single Flux Quantum (RSFQ), Energy Efficient Single Flux Quantum (eSFQ), and energy-efficient RSFQ (ERSFQ). Our formal model checking verification tool should theoretically work on RSFQ, eSFQ, and ERSFQ as well, but it has only been tested with standard SFQ circuits.

IV. Prior Works

Past research teams have created semi-formal and simulation-based tools for SFQ technology, as well as a tool for formal verification using Logical Equivalence Checking.

Tadros et al developed SystemVerilog HDL models for superconducting electronics that offer a general simulation and debugging platform[1]. These models captured the nature of the impact of circuit parameters, environmental effects, other phenomenon on superconducting circuits. HDL models before this for superconducting electronics were complicated and lacked modularity.

Wong et al improved upon this through designing a semi-formal verification framework for the SFQ circuits called VeriSFQ using UVM[2]. The VeriSFQ framework performs verification of SFQ logic through verifying the gate-level and circuit level properties. The VeriSFQ framework achieved speedup of verification for SFQ circuits and the development and created a benchmark for other SFQ verification methodologies to compare to.

Bakar et al developed a model checking verification framework, but it is not applicable to Single Flux Quantum technologies' unique properties [3]. They developed an automated formal verification technique that takes the design model and formal properties specifications as inputs and produces correctness results and counterexamples.

Krasniewski et al developed a library based on a representation of timing constraints in RSFQ circuits [4]. The library includes RSFQ cells as components within the library. They also performed logic simulation of RSFQ circuits and an RSFQ simulator was used for designing a decimation filter for a digital signal processing chip.

V. Key Novelties

We are improving on prior work through previous tools for SFQ technologies are semi-formal and simulation-based while our tool is a formal verification tool that improves both coverage and time of the verification. In summation we developed a model checking formal verification tool that can automatically generate testbench, expected results, and run both bounded and unbounded model checking in the Yosys tool. Our work produces a formal model checking verification framework for SFQ circuits, unlike prvious formal model checking tools for non-SFQ circuits.

VI. Framework

A. SFQ Verilog model

In semiconducting electronics, the binary information is represented in short quantized pulses called SFQ pulses. The SFQ basic convention is that the arrival of an SFQ pulse during the current clock period represents logic "1," whereas the absence of any pulse during this period is understood as logic "0" [2]. Verilog models of SFQ logic gates were developed simulating these SFQ-specific properties.The proposed models are publicly available on GitLab.

1) OR Gate: Figure 1 depicts the behavior of SFQ OR gate. It can be inferred from the waveform that the output pulls up when at least one input was given a SFQ pulse.

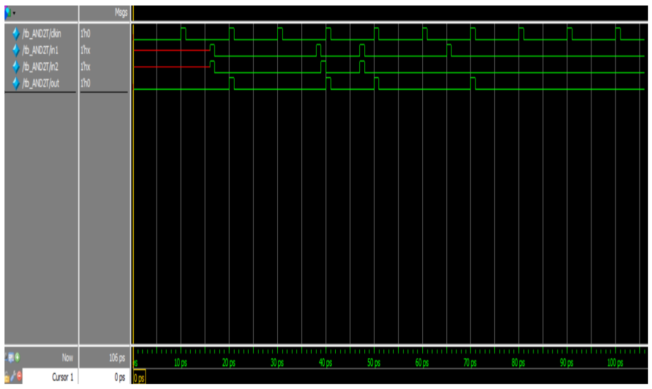


Fig. 1. SFQ OR Gate

2) AND Gate: Figure 2 illustrates the behavior of SFQ AND gate. It can be inferred from the waveform that the output pulls up only when both the inputs were given a SFQ pulse.

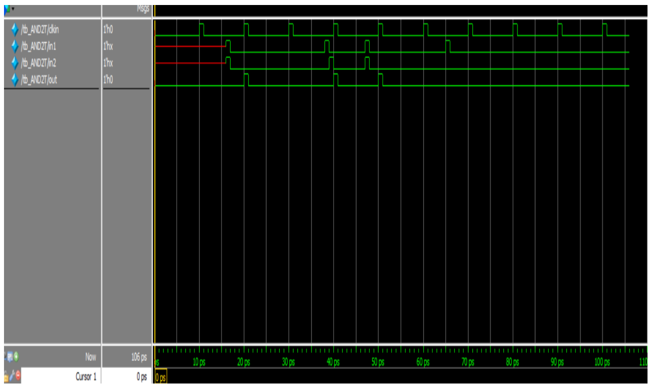


Fig. 2. SFQ AND Gate

3) XOR Gate: Figure 3 shows the behavior of SFQ XOR gate. It can be inferred from the waveform that the output pulls up only when any one of the input was given a SFQ pulse. The input, output, and clock are all pulses showing SFQ specific properties.

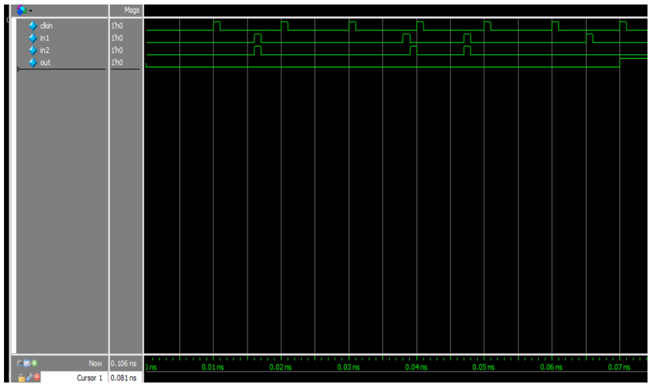


Fig. 3. SFQ XOR Gate

4) NOT Gate: Figure 4 shows the behavior of SFQ NOT gate. It can be inferred from the waveform that the output pulls up only when no input pulse was given.

5) DFF: Figure 5 shows the behavior of SFQ DFF. It can be inferred from the waveform that the output pulls up only if there exists an input SFQ pulse.

6) SPLITTER: Traditionally, SFQ circuits have a maximum fanout of one. When a gate requires more than one fanout, a special SFQ gate called Splitter is inserted at the output of that gate [1]. Figure 6 shows the behavior of SFQ Splitter. The input pulse is split and given to two outputs.

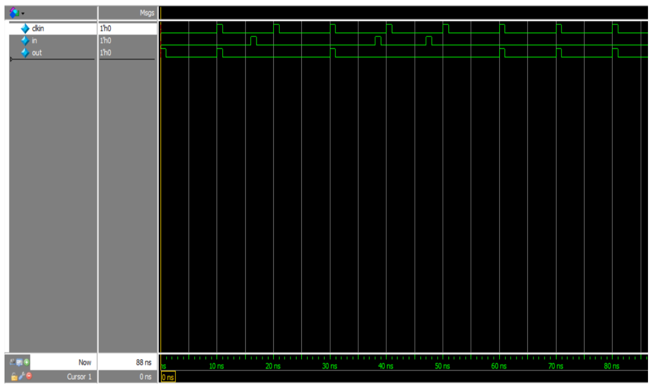


Fig. 4. SFQ NOT Gate

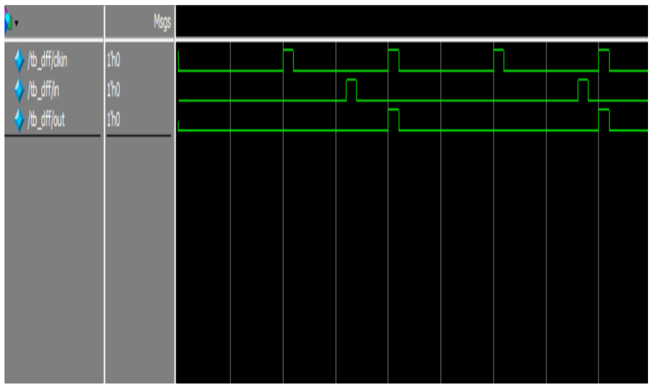


Fig. 5. SFQ DFF Gate

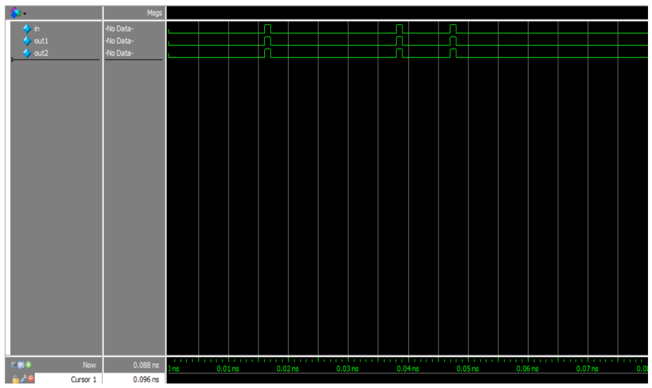


Fig. 6. SFQ SPLITTER

B. Model Checking

Model checking is an automated formal verification technique that takes formal system design model and formal properties specifications as input to perform verification and produce system correctness results and counterexamples [ref]. This technique is incorporated in our tool to verify the formal proofs and induction cases of the SFQ combinational circuits given as inputs in the form of a netlist to qMC. The formal proof of KSA is shown below as an example.

1) Formal Proof of Kogge-Stone Adder: KSAs are parallel prefix adders used for high performance arithmetic structures in modern computing.

Definition: Let $x = (x_{n-1}x_{n-2}...x_0)$ and $y = (y_{n-1}y_{n-2}...y_0)$ be the two N-bit inputs to an adder K. The output $sum = (c_n s_{n-1} s_{n-2}...s_0)$ of the adder K is formalized as follows:

$$P_i = x_i \oplus y_i \tag{1}$$

$$G_i = x_i \ . \ y_i \tag{2}$$

$$P_{(i:k)} = P_{(i:j)} \ . \ P_{(j-1:k)} \tag{3}$$

$$G_{(i:k)} = G_{(i:j)} + G_{(j-1:k)} \ . \ P_{(i:j)} \tag{4}$$

$$sum_i = P_i \oplus c_i \tag{5}$$

$$c_{i+1} = (P_i \ . \ c_0) + G_i \tag{6}$$

where, P_i - Propagate bit i
G_i - Generate bit i

C. qMC tool

Verification of circuits has been one of the major application areas of classical model checking.However, Model Checking applied to verification of quantum circuits is an area to be exploited systematically[ref]. Our qMC tool that we created verifies the design based on formal verification technique and it formally verifies the nature of the circuit based on formal assertions. It takes the input as a Berkeley Logic Interchange Format(blif) file that is generated through SFQMap synthesis tool [5]. It generates a SystemVerilog design file, configuration(config.sby) file and SystemVerilog testbench(tb) file. Then the generated files are provided to SymbiYosys Boundary Model Checking(BMC) that will further provide us with the PASS/FAIL information. Moreover, if there will be failure in the formal assertions, then the qMC tool will generate the counter-example and its corresponding waveform in GTKWave (trace.vcd file).

D. Flow Chart of Quantum Model Checking Tool (qMC Tool)

The flowchart in Figure 7 shows the overall workflow when using th qMC tool. The flow consists of using the SFQ Map synthesis tool to generate a blif file, which is sent to the qMC pre-processing translator. The pre-processing translator generates the SystemVerilog design file, testbench, and configuration file. Yosys BMC then takes these three files as inputs and outputs whether the formal assertions passed or failed. If they fail then a waveform of a counterexample is shown in GTKWave.

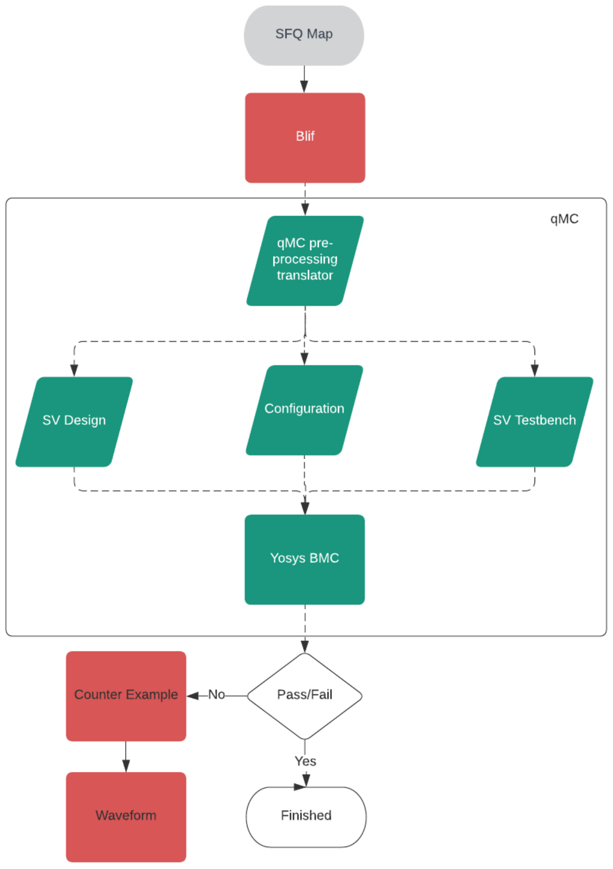


Fig. 7. Flow Chart

VII. Implementation

The qMC tool was verified through running the qMC tool on multiple Single Flux Quantum Kogge-Stone Adders. The Kogge-Stone Adder was then simulated in QuestaSim using its generated SystemVerilog design file and testbench file. The output of the Kogge-Stone Adder simulation was then compared to its expected output. We also verified the Kogge-Stone Adder through formal assertions that the tool automatically ran.

A. qMC Command

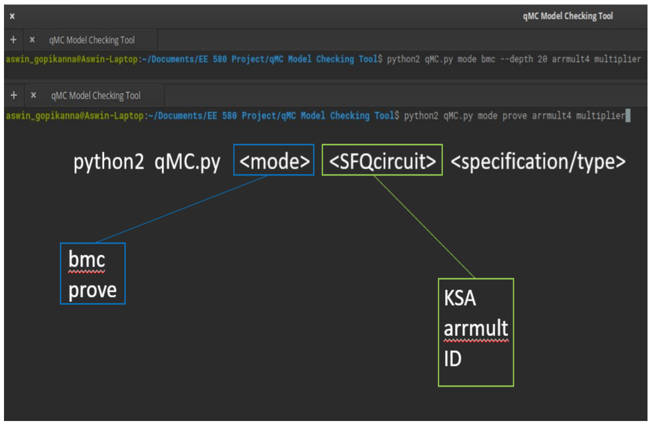
Figure 8 shows the command to run the qMC tool:

$$python\ qMC.py \ \ < Mode > < Blif > < Type > \tag{7}$$

where, $Mode$ specifies the model checking engine
$Blif$ is the input netlist file
$Type$ is the specification of the SFQ_circuit

B. Blif Inputs

The netlists of SFQ-based combinational circuits: KSA, ARM, ID and the counter were obtained from the SFQMap Synthesis Tool.



```
aswin_gopikanna@Aswin-Laptop:~/Documents/EE 580 Project/qMC Model Checking Tool$ python2 qMC.py mode bmc --depth 20 arrmult4 multiplier
aswin_gopikanna@Aswin-Laptop:~/Documents/EE 580 Project/qMC Model Checking Tool$ python2 qMC.py mode prove arrmult4 multiplier
```

```
python2 qMC.py <mode> <SFQcircuit> <specification/type>
```

Fig. 8. CMD

C. qMC Pre-Processing Translator

The qMC tool takes the input netlist file as an argument and processes it before running model checking. The pre-processing includes:-

1) blif to SV Design translation: The blif file is translated to a SystemVerilog design file constructed with SFQ logic gates. This enables the combinational circuits to simulate the SFQ-specific behavior.

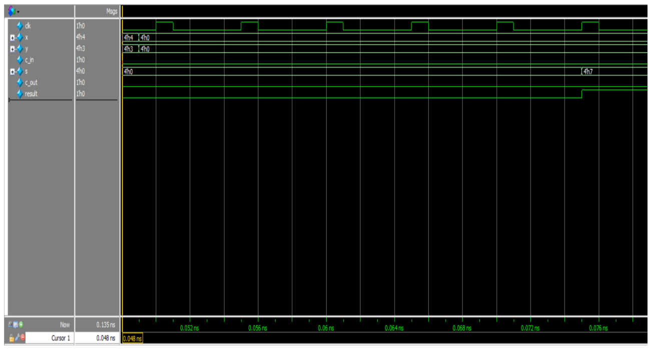


Fig. 9. KSA4 Waveform

KSA4:

KSA4 is a 4-bit Kogge-Stone Adder. The input of 4 and 3 along with the corresponding output of 7 can be seen in Figure 9. This shows the proper function of the SFQ circuit.

ARRMULT4:

ARRMULT4 is a 4-bit Array Multiplier. The input of 5 and 3 along with the corresponding output of 15 can be seen in Figures 10 and 11. This shows the proper function of the SFQ circuit.

ID4:

ID4 is a 4-bit Integer Divider. The input of 12 and 4 along with the corresponding output of 3 can be seen in Figures 12 and 13. This shows the proper function of the SFQ circuit.

COUNTER:

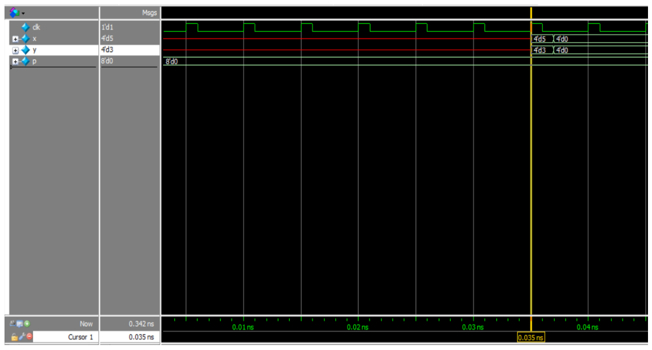


Fig. 10. ARRMULT4 Input Waveform

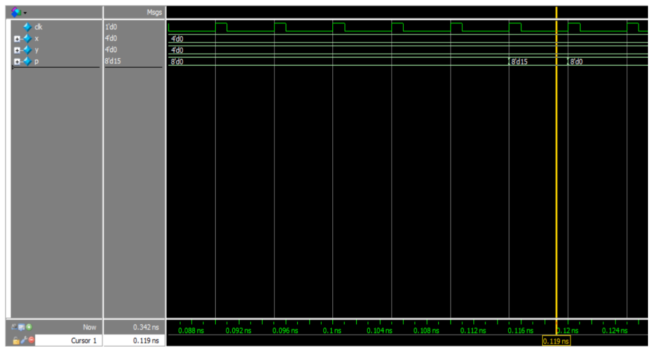


Fig. 11. ARRMULT4 Output Waveform

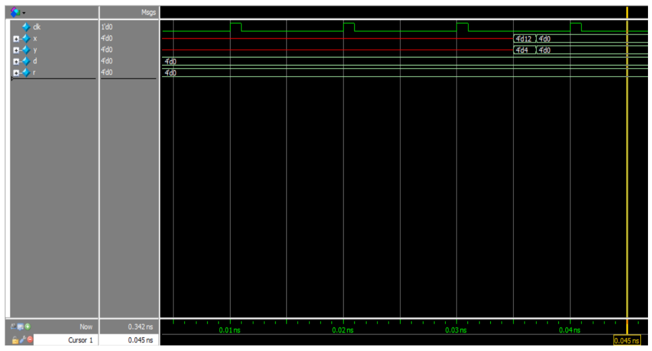


Fig. 12. ID4 Input Waveform

Counter output Waveform shown in Figure 14 is a 3 bit counter. This shows proper function of the SFQ circuit.

2) Configuration File Generation: The config file of Yosys is automatically generated corresponding to the mode of model checking and the SFQ_circuit.

3) SV Testbench Generation: qMC tool generates the SystemVerilog testbench with formal model checking based assertions supported by the smtbmc engine. These assertions verify the SFQ-specific properties of the circuit extracted from the netlist and produce system correctness results and counterexamples.

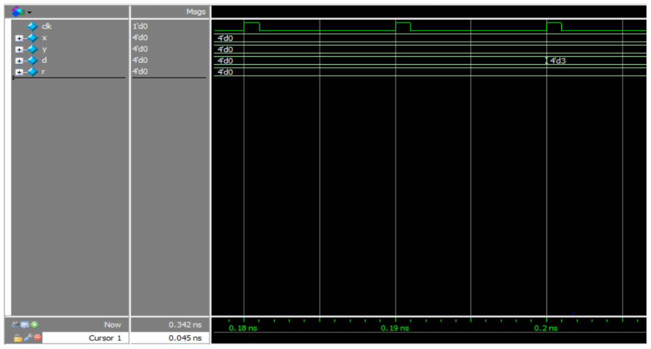


Fig. 13. ID4 Output Waveform

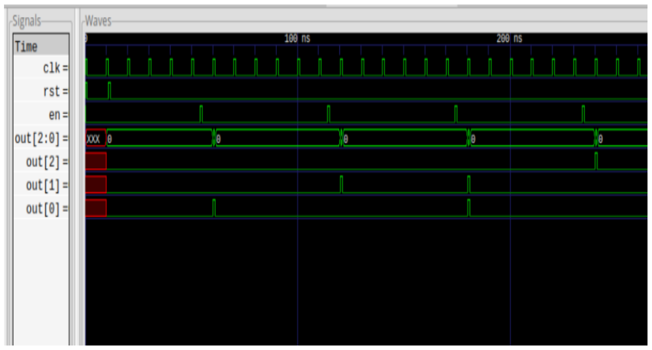


Fig. 14. Counter Output Waveform

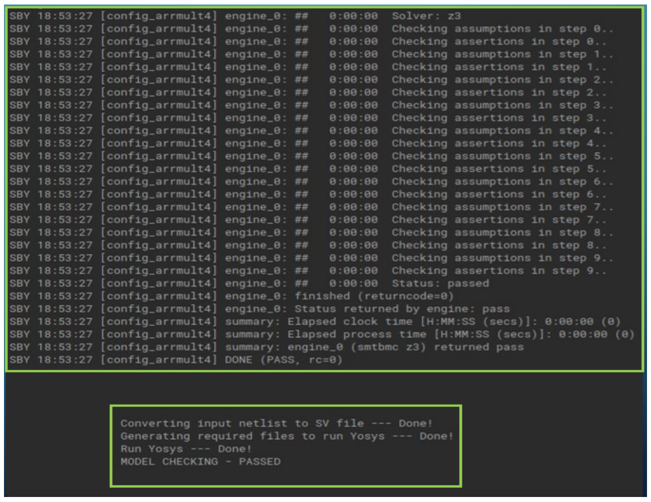
Algorithm 1 qMC Tool

```
1:  Let $Blif$ be the input netlist file of the SFQ based combinational circuit
2:  $qMC\_pre-processing\_translator$ ($Blif$)
3:  $Blif\_to\_SV\_translator$ (SFQ_circuit)
4:  for gate in SFQ_circuit do
5:      Instantiate $SFQ\_Verilog\_Model$ (gate)
6:  end for
7:  $Generate\_Testbench$ ($Blif$)
8:  Assert(SFQ_circuit)
9:  $Generate\_Config(Mode, Depth)$
10: if $Mode$ is BMC then
11:     Run bounded model checking based engine with depth equals $Depth$
12: end if
13: if $Mode$ is Prove then
14:     Run unbounded model checking based engine
15: end if
16: $Get\_Result$(result)
17: if result is FAIL then
18:     Display $Counterexample$ as waveform
19: else
20:     FINISH
21: end if
```

VIII. Results

A. PASS

The assertions are satisfied i.e. the formal proof and induction cases holds good. The depth of assertion is provided by the user as an argument.

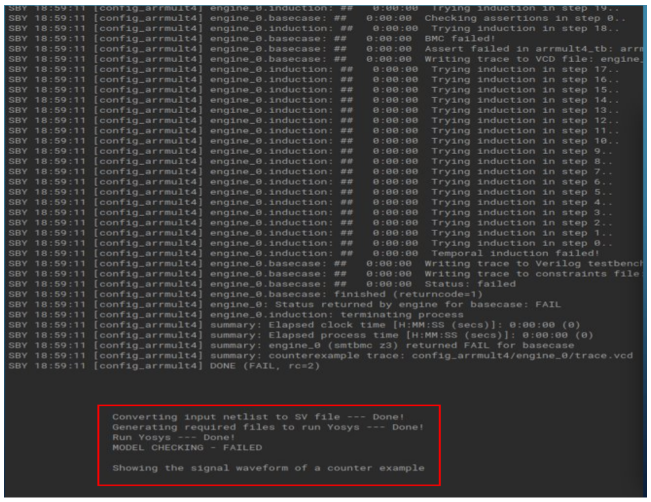


```
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Solver: z3
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 0..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 0..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 1..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 1..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 2..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 2..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 3..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 3..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 4..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 4..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 5..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 5..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 6..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 6..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 7..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 7..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 8..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 8..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assumptions in step 9..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Checking assertions in step 9..
SBY 18:53:27 [config_arrmult4] engine_0: ##   0:00:00  Status: passed
SBY 18:53:27 [config_arrmult4] engine_0: finished (returncode=0)
SBY 18:53:27 [config_arrmult4] engine_0: Status returned by engine: pass
SBY 18:53:27 [config_arrmult4] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 18:53:27 [config_arrmult4] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 18:53:27 [config_arrmult4] summary: engine_0 (smtbmc z3) returned pass
SBY 18:53:27 [config_arrmult4] DONE (PASS, rc=0)

        Converting input netlist to SV file --- Done!
        Generating required files to run Yosys --- Done!
        Run Yosys --- Done!
        MODEL CHECKING - PASSED
```

Fig. 15. PASS

B. FAIL

The MC based formal assertions failed as the induction cases are not met for the design specification given as input.



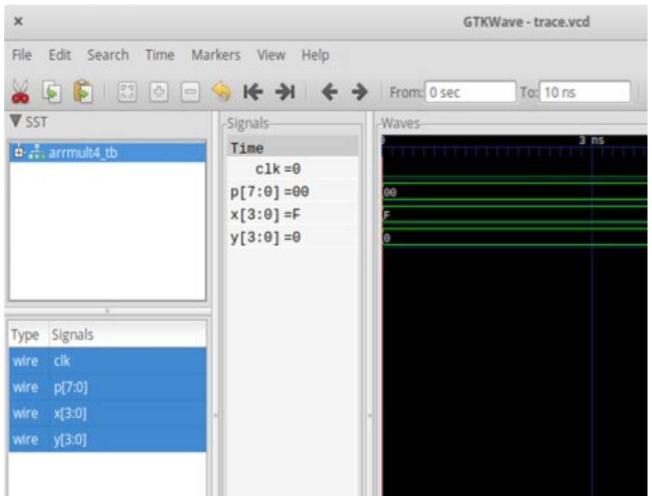
```
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 19..
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  Checking assertions in step 0..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 18..
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  BMC failed!
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  Assert failed in arrmult4_tb: arrm
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  Writing trace to VCD file: engine_
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 17..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 16..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 15..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 14..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 13..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 12..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 11..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 10..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 9..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 8..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 7..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 6..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 5..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 4..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 3..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 2..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 1..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Trying induction in step 0..
SBY 18:59:11 [config_arrmult4] engine_0.induction: ##   0:00:00  Temporal induction failed!
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  Writing trace to Verilog testbench
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  Writing trace to constraints file
SBY 18:59:11 [config_arrmult4] engine_0.basecase: ##   0:00:00  Status: failed
SBY 18:59:11 [config_arrmult4] engine_0.basecase: finished (returncode=1)
SBY 18:59:11 [config_arrmult4] engine_0: Status returned by engine for basecase: FAIL
SBY 18:59:11 [config_arrmult4] engine_0.induction: terminating process
SBY 18:59:11 [config_arrmult4] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 18:59:11 [config_arrmult4] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 18:59:11 [config_arrmult4] summary: engine_0 (smtbmc z3) returned FAIL for basecase
SBY 18:59:11 [config_arrmult4] summary: counterexample trace: config_arrmult4/engine_0/trace.vcd
SBY 18:59:11 [config_arrmult4] DONE (FAIL, rc=2)

        Converting input netlist to SV file --- Done!
        Generating required files to run Yosys --- Done!
        Run Yosys --- Done!
        MODEL CHECKING - FAILED

        Showing the signal waveform of a counter example
```

Fig. 16. FAIL

qMC generates and displays the counterexample waveform of the failed case.

IX. SMT

We also verified the counter using satisfiability modulo theories (SMT) in Yosys. To do this we used formal LTL assertions.



```
Time
   clk =0
p[7:0] =00
x[3:0] =F
y[3:0] =0
```

Type	Signals
wire	clk
wire	p[7:0]
wire	x[3:0]
wire	y[3:0]

Fig. 17. COUNTEREXAMPLE

X. Comparison With Baseline Approaches

Previous verification tools for SFQ logic have been simulation and semi-formal based. The qMC tool that we developed on the other hand is a formal model checking based tool that provides system correctness results and a waveform of a counterexample in the case of a failed assertion. Our tool improves both coverage and time of the verification through formal methodologies.

XI. ACKNOWLEDGEMENT

The authors would like to thank Dr. Shahin Nazarian, Professor at Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California. Thanks to the student advisor Arash Fayyazi, for his valuable inputs and guidance throughout the project.

References

[1] R. N. Tadros, A. Fayyazi, M. Pedram and P. A. Beerel, "SystemVerilog Modeling of SFQ and AQFP Circuits," in IEEE Transactions on Applied Superconductivity, vol. 30, no. 2, pp. 1-13, March 2020, Art no. 1300513, doi: 10.1109/TASC.2019.2957196.

[2] A. D. Wong, K. Su, H. Sun, A. Fayyazi, M. Pedram, and S. Nazarian, "VeriSFQ : A Semi-formal Verification Framework and Benchmark for Single Flux Quantum Technology," ISQED, pp. 224–230, 2019

[3] N. A. Bakar and A. Selamat, "Agent-based model checking verification framework," 2012 IEEE Conference on Open Systems, Kuala Lumpur, 2012, pp. 1-4.

[4] A. Krasniewski, "Logic simulation of RSFQ circuits," in IEEE Transactions on Applied Superconductivity, vol. 3, no. 1, pp. 33-38, March 1993, doi: 10.1109/77.233410.

[5] G. Pasandi, A. Shafaei, and M. Pedram, "SFQmap: A Technology Mapping Tool for Single Flux Quantum Logic Circuits," in 2018 IEEE Int. Symp. Circuits and Syst. (ISCAS), Florence, Italy, May 27-30, 2018, pp. 1-5.

[6] "W. N. N. Hung, Xiaoyu Song, Guowu Yang, Jin Yang and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no. 9, pp. 1652-1663, Sept. 2006, doi: 10.1109/TCAD.2005.858352."

[7] J. Frossl, J. Gerlach and T. Kropf, "An efficient algorithm for real-time symbolic model checking," Proceedings EDTC European Design and Test Conference, Paris, France, 1996, pp. 15-20, doi: 10.1109/EDTC.1996.494120.

[8] R. Li and Y. Liu, "Compositional Stochastic Model Checking Probabilistic Automata via Symmetric Assume-Guarantee Rule," 2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA), Honolulu, HI, USA, 2019, pp. 110-115, doi: 10.1109/SERA.2019.8886808.

[9] P. Camurati and P. Prinetto, "Formal verification of hardware correctness: introduction and survey of current research," in Computer, vol. 21, no. 7, pp. 8-19, July 1988, doi: 10.1109/2.65.

[10] A. Chhokra, S. Abdelwahed, A. Dubey, S. Neema and G. Karsai, "From system modeling to formal verification," 2015 Electronic System Level Synthesis Conference (ESLsyn), San Francisco, CA, 2015, pp. 41-46.

[11] A. S. Wojcik, "Formal Design Verification of Digital Systems," 20th Design Automation Conference Proceedings, Miami Beach, FL, USA, 1983, pp. 228-234, doi: 10.1109/DAC.1983.1585653.

[12] A. M. Haslam, K. M. English, A. Derrickson and J. F. McDonald, "Automated Verification and Optimization of SFQ Superconducting Circuits," in IEEE Access, vol. 7, pp. 22843-22855, 2019, doi: 10.1109/ACCESS.2019.2899873.

[13] N. K. Katam, J. Kawa and M. Pedram, "Challenges and the status of superconducting single flux quantum technology," 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 2019, pp. 1781-1787, doi: 10.23919/DATE.2019.8747356.

[14] L. Ji, J. Ma and Z. Shan, "Research on Model Checking Technology of UML," 2012 International Conference on Computer Science and Service System, Nanjing, 2012, pp. 2337-2340, doi: 10.1109/CSSS.2012.580.

[15] Katam, Naveen Pedram, Massoud. (2019). Timing Characterization for Static Timing Analysis of Single Flux Quantum Circuits. IEEE Transactions on Applied Superconductivity. PP. 1-1. 10.1109/TASC.2019.2891166.

[16] Katam, Naveen Shafaei, Alireza Pedram, Massoud. (2017). Design of Complex Rapid Single-Flux-Quantum Cells with Application to Logic Synthesis. 1-3. 10.1109/ISEC.2017.8314236.

[17] Tang, Guang-Ming Qu, Pei-Yao Ye, Xiao-Chun Fan, Dong-Rui. (2018). Logic Design of a 16-bit Bit-Slice Arithmetic Logic Unit for 32-/64-bit RSFQ Microprocessors. IEEE Transactions on Applied Superconductivity. PP. 1-1. 10.1109/TASC.2018.2799994.

[18] Katam, Naveen Pedram, Massoud. (2018). Logic Optimization, Complex Cell Design and Retiming of Single Flux Quantum Circuits. IEEE Transactions on Applied Superconductivity. PP. 1-1. 10.1109/TASC.2018.2856833.

[19] O. A. Mukhanov, "Energy-Efficient Single Flux Quantum Technology," in IEEE Transactions on Applied Superconductivity, vol. 21, no. 3, pp. 760-769, June 2011, doi: 10.1109/TASC.2010.2096792.

[20] A. Inamdar, D. Amparo, B. Sahoo, J. Ren and A. Sahu, "RSFQ/ERSFQ Cell Library With Improved Circuit Optimization, Timing Verification, and Test Characterization," in IEEE Transactions on Applied Superconductivity, vol. 27, no. 4, pp. 1-9, June 2017, Art no. 1302109, doi: 10.1109/TASC.2017.2677418.

[21] S. Polonsky, P. Shevchenko, A. Kirichenko, D. Zinoviev and A. Rylyakov, "PSCAN'96: new software for simulation and optimization of complex RSFQ circuits," in IEEE Transactions on Applied Superconductivity, vol. 7, no. 2, pp. 2685-2689, June 1997, doi: 10.1109/77.621792.

[22] C. A. Hamilton and K. C. Gilbert, "Margins and yield in single flux

quantum logic," in IEEE Transactions on Applied Superconductivity, vol. 1, no. 4, pp. 157-163, Dec. 1991, doi: 10.1109/77.107400.

[23] Serna-M and David, 2014, Edgar Serna-M., Morales-V. David "State of the Art in the Research of Formal Verification" Ingenieria Investigacion y Tecnologia, XV (04) (2014), pp. 615-623

[24] Batra, Mona, Amit Malik and Meenu Dave. "FORMAL METHODS: BENEFITS, CHALLENGES AND FUTURE DIRECTION." Journal of Global Research in Computer Sciences 4 (2013): 21-25.

[25] C. J. Fourie and M. H. Volkmann, "Status of Superconductor Electronic Circuit Design Software," in IEEE Transactions on Applied Superconductivity, vol. 23, no. 3, pp. 1300205-1300205, June 2013, Art no. 1300205, doi: 10.1109/TASC.2012.2228732.

[26] C. Fourie, "Single Flux Quantum Circuit Technology and CAD overview," 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, 2018, pp. 1-6, doi: 10.1145/3240765.3243498.

[27] K. K. Likharev and V. K. Semenov, "RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems," in IEEE Transactions on Applied Superconductivity, vol. 1, no. 1, pp. 3-28, March 1991, doi: 10.1109/77.80745.

[28] O. A. Mukhanov, S. V. Rylov, V. K. Semonov and S. V. Vyshenskii, "RSFQ logic arithmetic," in IEEE Transactions on Magnetics, vol. 25, no. 2, pp. 857-860, March 1989, doi: 10.1109/20.92421.

[29] D. E. Kirichenko, S. Sarwana and A. F. Kirichenko, "Zero Static Power Dissipation Biasing of RSFQ Circuits," in IEEE Transactions on Applied Superconductivity, vol. 21, no. 3, pp. 776-779, June 2011, doi: 10.1109/TASC.2010.2098432.

[30] H. Hayakawa, N. Yoshikawa, S. Yorozu and A. Fujimaki, "Superconducting digital electronics," in Proceedings of the IEEE, vol. 92, no. 10, pp. 1549-1563, Oct. 2004, doi: 10.1109/JPROC.2004.833658.

[31] R. N. Tadros and P. A. Beerel, "A Robust and Self-Adaptive Clocking Technique for SFQ Circuits," in IEEE Transactions on Applied Superconductivity, vol. 28, no. 7, pp. 1-11, Oct. 2018, Art no. 1301211, doi: 10.1109/TASC.2018.2856836.

[32] Kris Gaj, Chin-Hong Cheah, E. G. Friedman and M. J. Feldman, "Functional modeling of RSFQ circuits using Verilog HDL," in IEEE Transactions on Applied Superconductivity, vol. 7, no. 2, pp. 3151-3154, June 1997, doi: 10.1109/77.622000.

[33] N. Yoshikawa and J. Koshiyama, "Top-down RSFQ logic design based on a binary decision diagram," in IEEE Transactions on Applied Superconductivity, vol. 11, no. 1, pp. 1098-1101, March 2001, doi: 10.1109/77.919539