qMC: A Formal Model Checking Verification Framework For Superconducting Logic

Mustafa Munir
Ming Hsieh Department of Electrical
and Computer Engineering
University of Southern California
Los Angeles, California, USA
mmunir@usc.edu

Aswin Gopikanna
Ming Hsieh Department of Electrical
and Computer Engineering
University of Southern California
Los Angeles, California, USA
gopikann@usc.edu

Arash Fayyazi
Ming Hsieh Department of Electrical
and Computer Engineering
University of Southern California
Los Angeles, California, USA
fayyazi@usc.edu

Massoud Pedram
Ming Hsieh Department of Electrical
and Computer Engineering
University of Southern California
Los Angeles, California, USA
pedram@usc.edu

Shahin Nazarian
Ming Hsieh Department of Electrical
and Computer Engineering
University of Southern California
Los Angeles, California, USA
shahin.nazarian@usc.edu

Abstract

Single flux quantum (SFQ) circuits as an example of superconducting electronics (SCE) have the potential to replace CMOS circuits as they possess a theoretical potential of three orders of magnitude reduction in power accompanied with one order of magnitude higher speed. Despite its benefits, the SCE community lacks a reliable open source formal verification solution. This paper proposes a verification framework called qMC, a model checker for SFQ circuits using formal techniques. qMC offers an automated process 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). Instead of creating an MC tool from scratch, we have built qMC based on well established open source back-end verification engines for MC of CMOS circuits, including Yosys-SMTBMC and EBMC. qMC allows for properties to be given in SystemVerilog formal assertions, time-limited SystemVerilog assertions, or linear temporal logic (LTL). qMC provides an improvement in terms of verification time and coverage when compared to state-of-the-art semi-formal based SFQ verification frameworks. For instance, verification time for a 4-bit array multiplier is sped up by 19.5x.

The research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office grant W911NF-17-1-0120. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation herein. This project is also supported in part by a grant from the Software and Hardware Foundations program of the National Science Foundation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions.acm.org.

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8393-6/21/06...\$15.00

<https://doi.org/10.1145/3453688.3461522>

CCS Concepts

• **Hardware** → **Emerging technologies**; **Model checking**; **Quantum technologies**.

Keywords

Single-Flux Quantum (SFQ), Model Checking (MC), Formal Verification, System Verification, Verilog, SystemVerilog

ACM Reference Format:

Mustafa Munir, Aswin Gopikanna, Arash Fayyazi, Massoud Pedram, and Shahin Nazarian. 2021. qMC: A Formal Model Checking Verification Framework For Superconducting Logic. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21)*, June 22–25, 2021, Virtual Event, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3453688.3461522>

1 Introduction

The end of Moore's Law has led to a desire to find alternative technologies that can replace the prevalent use of complementary metal oxide semiconductor (CMOS) transistors [21, 25]. Currently CMOS technologies have a vast set of design and verification tools available in industry. For other technologies to become viable, a major shortcoming to address would be the lack of robust verification techniques and tools [2, 16]. Despite the challenges other technologies face there are still some promising technologies like superconducting electronics (SCE) [9, 20].

For new technologies to compete with CMOS, they must be low power, high speed, and have the necessary electronic design automation (EDA) tools [13]. A type of SCE called single flux quantum (SFQ) technology meets this need for low power consumption and high operating speed, but does not yet have all the necessary EDA tool support [14, 19]. Note that SFQ circuits operate at cryogenic temperatures and have a reduction in power consumption of three orders of magnitude when compared to high-end CMOS circuits, while still providing an improvement in speed [11, 21].

SFQ circuits are all sequential circuits in order to capture the pulses given as inputs. The sequential nature of SFQ circuits makes developing formal verification frameworks for SFQ circuits more difficult because even simple combinational designs like a half adder

are a sequential circuit in SFQ technology. Since every gate has its own states [15], the verification of combinations of such states can lead to state explosion.

Researchers have created semi-formal and simulation based tools for SFQ technology. Tadros et al. [21] developed SystemVerilog HDL models for SCE that offer a general simulation and debugging platform. Wong et al. [25] improved upon this through designing a semi-formal verification framework for the SFQ circuits called VeriSFQ using universal verification methodology (UVM). The VeriSFQ framework performs verification of SFQ logic through verifying the gate-level and circuit-level specifications. The VeriSFQ framework achieved speedup of verification for SFQ circuits and created a benchmark for other SFQ verification methodologies to compare to.

The lack of strong verification tools for SFQ technology makes it 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 [6, 7]. As SFQ technology matures the development of EDA tools and robust verification tools becomes increasingly crucial [3, 18]. We developed qMC, an open source MC based formal verification framework for superconducting logic that can automatically generate a test-bench and run MC. qMC is built based on well established back-end engines for MC of CMOS circuits such as EBMC [17] and Yosys-SMTBMC [23]. qMC can run MC for a limited number of generated steps in a finite state machine to check whether a property violation occurred within the defined bound (bounded MC) or run unbounded MC, where the property violation is checked in all possible states. qMC also performs checks on the structural properties of SFQ circuits through our qMC pre-processing translator that is presented in section 3.1.

The key contributions of this work can be summarized as follows:

- (1) To the best of our knowledge, qMC is the first MC based formal verification framework for superconducting logic. qMC significantly improves verification quality in terms of coverage and verification time compared to previous semi-formal based verification frameworks, like VeriSFQ. qMC improves verification time by approximately 19.5x for a 4-bit array multiplier when compared to VeriSFQ [25]. qMC provides an improvement in coverage through the nature of formal verification providing 100% coverage of the assertions specified. qMC also produces system correctness results and counterexamples for superconducting circuits that are given as inputs in the form of a netlist to qMC.
- (2) qMC is an open source formal verification framework that can verify SFQ circuits, while other current CMOS formal verification tools are unable to verify SFQ circuits, due to their sequential nature, path balancing, and fanout. The sequential nature is handled in qMC through evaluating assertions after the delay caused by the input pulses propagating to the output of the circuit. The delay is handled through storing the input pulses into reg variables in Verilog. The storing of the input pulses enables us to record the values of the input pulses thereby allowing us to use the previous values of the input for an assertion that evaluates when the output pulse is generated. This takes

advantage of Past LTL (PLTL) discussed in [22] through allowing us to express properties on previously generated pulses. After the delay the output pulse is generated and we still have a record of whether the corresponding input pulse was previously generated or not, allowing us to handle SFQ's sequential nature.

- (3) qMC also supports time-limited SystemVerilog Assertions, SystemVerilog formal assertions, and LTL operations using the enhanced bounded model checker back-end engine for MC [17]. qMC's generated assertions also take advantage of metric temporal logic as the output has to be evaluated at a specific moment in time after the delay of the SFQ gates is propagated to the output. This is done through the use of SystemVerilog formal assertions like `s_eventually` as well as time-limited assertions to evaluate the output after a specified delay. This deals with the issue of representing a circuit with real-time constraints as discussed in the timed automata sections of [4].
- (4) We designed Verilog models that capture the pulse-based sequential nature of SFQ and are compatible with open source tools for qMC to use.

While the scope of this work is focused on RTL MC, we also validate the accuracy of our proposed Verilog models similarly to [21] where the JSIM [5] and Verilog simulations match digitally with the outputs appearing at the same cycles. The qMC implementation is publicly available on GitLab [1]. The remainder of the paper is organized as follows. Section 2 discusses the proposed Verilog models for SFQ technology. Then, Section 3 describes qMC in detail while Section 4 studies the effectiveness of the proposed verification framework. Finally, Section 5 presents our conclusions.

2 qMC Verilog Model and Properties

In SCE, 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" [25]. Verilog models of SFQ logic gates were developed simulating these SFQ-specific properties for qMC. The SFQ gates designed all use temporary state registers which store the presence or absence of the SFQ pulse, thereby preserving the pulse-based nature of the SFQ technology while also retaining the information from the input pulses. The SFQ Verilog design files developed are used as input files to the back-end engines for MC [17, 23] in qMC to perform Satisfiability Modulo Theories (SMT) based MC. SMT based MC verifies that the required specifications of the model are met within all possible states of the given model.

SFQ logic can only drive one other node, limiting the fanout to one [20]. When an SFQ gate needs to drive more than one node then a special SFQ gate called a splitter has to be used at the output of the gate to make the fanout two [20]. SFQ gates also require path balancing for correct operation [20]. Due to gate-level pipelining in SFQ, all inputs of a gate should have an equivalent length of the longest path (in terms of the gate count) [20]. For path balancing, D Flip-Flops (DFF) should be added to the path with the shorter length of longest path [20]. Path balancing is necessary in SFQ circuits because when the clock arrives at a gate the input value is reset in order to produce the output value meaning that if an input violates setup or hold time then the gate will produce incorrect results when the clock pulse arrives [20]. Path balancing and fanout

checks are handled in the pre-processing translator of qMC through the path balancing checker and fanout checker that are discussed in the qMC pre-processing translator Section 3.1 [25].

2.1 SFQ Gates

The SFQ gates that were modeled are the OR gate, AND gate, XOR gate, NOT gate, D Flip-Flop, and Splitter. The behavior of the SFQ OR gate is that whenever one input receives a pulse the output generates a pulse in the following clock cycle. It can be inferred from the gate model in Fig. 1 that the output pulse is generated when at least one input was given an SFQ pulse. When neither input receives a pulse then there is no output pulse generated as seen in the second assertion in Fig. 1. Since the circuit is an SFQ circuit it behaves in a sequential fashion and the output changes at the edge of the clock. The delay of one cycle for the output to become high is the result of the accumulated delay from both of the input pulses having their values stored into in1_state and in2_state. This delay is present in all of the SFQ gates designed, except for the Splitter. The inputs, outputs, and clock are all pulses in all of the SFQ gates showing SFQ-specific properties.

```

OR2T(input clk, input in1, input in2, output reg out);
  reg in1_state;
  reg in2_state;
  initial in1_state = 0;
  initial in2_state = 0;
  always @(posedge in1 or posedge in2 or posedge clk)
    if(in1)
      in1_state = in1 | in1_state;
    if(in2)
      in2_state = in2 | in2_state;
    else if (clk)
      in1_state <= 0;
      in2_state <= 0;
  always @(posedge clk)
    out <= in1_state | in2_state;
    out <= #1 1'b0;
  assert property (in1 == 1 | in2 == 1 |>=> ##1 out == 1);
  assert property (in1 == 0 & in2 == 0 |>=> ##1 out == 0);

```

Figure 1: Verilog modeling of SFQ OR gate.

Fig. 2 depicts the waveform of the SFQ-based OR gate. It can be seen that all of the input and output signals behave in a pulse-like manner and that there is a delay of one time unit from the input to the output. This waveform matches the QuestaSim simulation of the Verilog Modeling for the SFQ OR Gate.

Figure 2: Waveform showing the pulse-based operation of an SFQ OR gate.

Fig. 3 illustrates the assertions which check the properties of the individual SFQ gates, which are listed in a comment that is represented by "//". For brevity, the cases where an output pulse is not generated are not shown for the SFQ gates. In the case of the XOR gate, it can be inferred from the assertions that the output pulse is generated when any one of the inputs was given an SFQ pulse. The remaining SFQ gates logic can be seen in Fig. 3. A special SFQ gate called a Splitter is inserted at the output of a gate [21] when a gate requires a fanout of more than one. A splitter is needed because traditionally, SFQ circuits have a maximum fanout of one. For the DFF it can be inferred from the assertion that the output pulse is generated if there exists an input SFQ pulse.

```

1 assert property SFQ_Gate_Assertions:
2   (in1 == 1 & in2 == 1 |>=> ##1 out == 1); // AND
3   (in1 != in2 |>=> ##1 out == 1); // XOR
4   (in1 == in2 |>=> ##1 out == 0); // XOR
5   (in1 == 1 |>=> ##1 out == 1); // DFF
6   (in1 == 0 |>=> ##1 out == 1); // NOT
7   (in1 == 1 |>=> out1 == 1 && out2 == 1); //Splitter
8 endproperty

```

Figure 3: SFQ Gate assertion properties where a comment is represented by "//".

3 qMC: Our Proposed Framework

Verification of circuits has been one of the major application areas of classical MC. However, MC applied to the verification of quantum circuits is an area to be exploited systematically [8]. Our qMC framework verifies the nature of the design through the use of MC to verify the specified formal assertions. The flowchart in Fig. 4 shows the overall workflow when using the qMC verification framework. The flow consists of using the SFQMap synthesis tool [20] 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 (config.sby) file. The generated files are then provided to the back-end engine for MC [17]. qMC then takes these three files as inputs and outputs whether the formal assertions passed or failed. If they fail then a counterexample is saved in the form of a VCD file. qMC was coded in Python 3 and verified through running on multiple SFQ Kogge-Stone Adders, Array Multipliers, and basic SFQ gates (OR, AND, XOR, NOT, DFF, SPLITTER). The circuits were simulated in QuestaSim using the generated SystemVerilog design file and testbench file. The output of the circuit simulation was then compared to its expected output. The netlists of SFQ-based circuits: KSA, ARRMULT, ID and the counter were obtained from the SFQMap Synthesis Tool [20].

3.1 qMC Pre-Processing Translator

The qMC tool takes the input netlist as an argument and processes it before running MC. The pre-processing includes: blif to SystemVerilog design translation, configuration file generation, and testbench generation. The blif file is translated to a SystemVerilog design file constructed with SFQ logic gates. This enables the SystemVerilog circuits to simulate the SFQ-specific behavior. The config file of the back-end engine for MC is automatically generated corresponding to the mode of MC and the SFQ circuit. qMC also generates the SystemVerilog testbench with formal MC based assertions or time-limited assertions based on the input blif file. These assertions verify

Figure 4: The flowchart of our formal SFQ verification framework, qMC. The dotted line shows the portion of the framework that encompasses qMC.

the SFQ-specific properties of the circuit extracted from the netlist and produce system correctness results and counterexamples.

In the pre-processing translator of qMC there is a fanout checker and a path balancing checker. The pseudocode in Algorithm 1 shows the basic operation of the fanout checker and the path balancing checker. The fanout checker adjusts the SFQ gate fanout when splitters are inserted for any gate output connected to two other gates. The path balancing checker assumes the SFQ circuit is represented as a Directed Acyclic Graph (DAG). It uses Depth First Search (DFS) to visit all the nodes in the DAG to ensure path balancing as seen in the PBCheckerDFS function in Algorithm 1 that returns a 1 if path is balanced and 0 otherwise [25]. Upon passing both checkers, the structural model is converted into the SystemVerilog design. If the design does not pass both of the checkers then, the design is flagged for the corresponding error. Flagged designs are rejected from undergoing verification in qMC, thus saving verification time [25].

Algorithm 1 Fanout and Path Balancing Checker Pseudocode.

```

1: for each Primary Input do
2:   if Fanout != 1 then
3:     return Fanout Error
4:   end if
5: end for
6: for each Gate do
7:   if Gate == Splitter then
8:     if Fanout != (1 or 2) then
9:       return Fanout Error
10:    end if
11:  else
12:    if Fanout != 1 then
13:      return Fanout Error
14:    end if
15:  end if
16: end for
17: IsPathBalanced = PBCheckerDFS(nodes)
18: if IsPathBalanced == 0 then
19:   return Path Balancing Error
20: else
21:   return No Error
22: end if
  
```

3.2 Back-End Engines for Model Checking

The developed qMC tool is able to use two different back-end engines for MC: Yosys-SMTBMC and EBMC.

3.2.1 Yosys-SMTBMC Yosys-SMTBMC is a free and open source tool with a focus on verification using bounded MC (BMC) and k-induction [23, 24]. Yosys-SMTBMC uses the SMT2 circuit descriptions that are generated by Yosys when it is provided a Verilog design file with assertions [24]. When the MC is performed the process can return a pass meaning that no reachable state from the initial state violates the specified assertions [24]. It can also return a fail when there is a state reachable within the user specified k steps from the initial state that violates some of the specified assertions. A counter-example trace is produced in VCD format when a fail occurs [24]. When Yosys is used as the back-end engine for MC, the qMC tool does not have the ability to support time-limited SystemVerilog assertions or LTL operations since Yosys does not support these. Thus, when Yosys is used as the back-end engine for qMC, it supports formal assertions in SystemVerilog.

3.2.2 EBMC EBMC takes a Verilog, SystemVerilog, or SMV design file with assertions as an input and uses BMC or k-induction to perform SMT based MC [10, 17]. EBMC can also use the input Verilog design file with assertions to create a Boolean-level MC problem in SMV format [10]. It is also able to handle memories without abstracting them and features a BDD engine (unbounded) to perform MC [10]. When EBMC performs MC it returns a pass or fail with a counter-example and optional trace. Properties in EBMC can be given in LTL or a fragment of SystemVerilog assertions, including time-limited SystemVerilog assertions.

4 Results

In this section, the efficacy of qMC is investigated. First, the experimental setup and the considered combinational SFQ circuits are

described in section 4.1. Then, the expected waveforms of some of the combinational SFQ circuits are shown in section 4.2. In section 4.3, the runtime of qMC when running the benchmark circuits is considered. Finally, example assertions that are considered for the benchmark circuits are discussed and figures showing states of the counterexample trace for the KSA4 are shown.

4.1 Experimental Setup

The computer system used for the testing utilized an Intel Core i7-8550U CPU with a nominal clock frequency of 1.8 gigahertz and 12 gigabytes of RAM. The proposed qMC framework was evaluated under circuit designs to indicate the effectiveness of the proposed checkers and model. We used three different types of circuits, 4-, 8-, 16-, 32-bit KSAs, ARRMULTs, and IDs to verify the correct functionality using qMC. Since our framework's objective is to verify the correct functionality of the post-synthesis netlist, we used [20] as a synthesis tool to generate an equivalent gate-level structural model from a given circuit. Two types of design errors, namely functional and structural errors are inserted into the netlist to evaluate the effectiveness of the qMC framework. Functional errors are brought about by changing the functionality of the circuit by swapping gates. Structural errors are inserted to make the circuit partially-balanced by either deleting a DFF or by removing a Splitter to increase the fanout of a gate. The assertions capture these errors as the design given as an input netlist does not meet the design specification. qMC generates the counterexample and creates a .vcd file of the failed case.

4.2 SFQ Benchmark Circuits

Circuits such as KSA, ARRMULT and ID were developed using the typical SFQ gates introduced in earlier sections. The functional waveforms of ARRMULT4 are shown in Fig. 5. The inputs 7 and 11 to the array multiplier along with the corresponding output of 77 can be seen in Fig. 5. The delay for the 4-bit array multiplier in Fig. 5 is 16 time units since the delay of each SFQ gate propagates to the output. The line between clock pulse three and sixteen in Fig. 5 represents that the figure does not show the clock cycles in between to limit the size of the figure. The results waveform for the 4-, 8-, 16-, 32-bit KSA, 8-, 16-bit ARRMULT and 4-, 8-bit integer divider have been derived but are not included in this paper for brevity.

4.3 Verification Results

Table 1 shows the runtime in seconds, the depth, and the number of gates of the benchmarks that were run on qMC. To compare to qMC, Table 2 shows the runtime in seconds and the speedup when compared to VeriSFQ when running the ARRMULT benchmarks on VeriSFQ [25]. For brevity, only ARRMULT benchmark times are shown in Table 2. The SFQ circuits in SFQ circuit benchmarks on [12] were used for the testing of qMC and VeriSFQ. From Table 2 it can be seen that qMC has a significant speedup compared to VeriSFQ even on larger circuits such as, the 16-bit array multiplier.

qMC verifies the correct functionality of a given netlist using assertions, which check the properties that are derived from the design specification. Assertions are said to be satisfied when the formal proof and induction cases hold true for the given bound. Fig. 6 shows the assertion which checks the property of the SFQ

Figure 5: ARRMULT4 waveforms. The arrow shows the delay from the inputs to the output is sixteen cycles. The line circled shows the thirteen cycles skipped for brevity.

Table 1: Runtime of qMC on KSA and ID benchmarks.

Benchmark	Depth	Runtime (s)	# of gates
KSA 4	50	1	87
KSA 8	50	2	230
KSA 16	50	3	592
KSA 32	50	9	1486
ID 4	50	3	540
ID 8	50	15	3175

Table 2: Comparison of VeriSFQ and qMC Runtimes on array multiplier benchmarks.

Benchmark	VeriSFQ (s)	qMC (s)	Speedup
ARRMULT 4	39	2	19.5x
ARRMULT 8	96	8	12x
ARRMULT 16	119	35	3.4x

KSA4 circuit. The sum of the KSA4 is delayed by six cycles as the input pulses propagate through the SFQ gates. Fig. 6 illustrates an example of the time-limited SystemVerilog assertions used, where reg_cout6 and reg_s6 represent the cout and sum propagated to their respective 6th register to store the value of the SFQ pulse.

```

1 assert property (x != 0 || y != 0 || c_in != 0
2 |> ##6 result == {reg_cout6, reg_s6});

```

Figure 6: SFQ KSA4 circuit assertion.

Fig. 7 illustrates the assertion which checks the property of the SFQ ARRMULT4 circuit. The product is delayed as the input pulses propagate through the SFQ gates, thus registers are used to store the values from the input pulses as explained in the discussion of KSA4. Fig. 7 shows an example of the SystemVerilog formal assertions used. The SFQ circuit assertions are available on GitLab [1].

Fig. 8 shows the states of a possible counterexample trace of KSA4 when the design is not functioning correctly and a counterexample is generated. The output generated after the delay of 6 states in Fig. 8 is incorrect, thus the trace shown is the counterexample. The solid lines represent moving to the next state and the dashed lines represent that the states in between are not shown for brevity. In Fig. 8 the input variables x , y , and cin can be seen in each state from state 0 to state 7, and the corresponding output (sum and cout) generated after a delay of 6 states. If the output generated after the delay of 6 states in Fig. 8 was correct then it would be an example of a proper execution trace.

```
assert property (s_eventually product == reg_x17 * reg_y17);
```

Figure 7: SFQ ARRMULT4 circuit assertion.

Figure 8: Possible counterexample trace of KSA4 showing the incorrect output generated after the expected six cycle delay. The states show the input values propagate to the output after the delay due to the SFQ pulse propagating through the gates of the KSA4.

5 Conclusion

qMC is the first MC based formal verification framework for superconducting logic, while other current CMOS formal verification tools are unable to verify SFQ circuits. qMC also provides system correctness results and a counterexample trace in the case of a failed assertion. qMC provides an improvement in verification quality in terms of coverage and verification time through formal methodologies. When comparing qMC with VeriSFQ, a maximum speedup of 19.5x can be seen in verification time showing an improvement over current verification frameworks for SFQ technology. An improvement in coverage can also be seen through qMC providing 100% coverage of the assertions specified. qMC is also capable of handling formal assertions and time-limited assertions in SystemVerilog, or LTL operations. Future work includes expanding the qMC framework to operate on sequential circuits.

References

- [1] 2021. *qMC Source Code*. <https://gitlab.com/arash1902/qmc>
- [2] Najwa Abu Bakar and Ali Selamat. 2012. Agent-based model checking verification framework. (2012), 1–4. <https://doi.org/10.1109/ICOS.2012.6417649>
- [3] M. Batra, A. Malik, and Meenu Dave. 2013. Formal Methods: Benefits, Challenges and Future Direction. *Journal of Global Research in Computer Sciences* 4 (2013), 21–25.
- [4] Patricia Bouyer. 2009. Model-checking Timed Temporal Logics. *Electronic Notes in Theoretical Computer Science* 231 (2009), 323–341. <https://doi.org/10.1016/j.entcs.2009.02.044> Proceedings of the 5th Workshop on Methods for Modalities.
- [5] E. Fang and T. Van Duzer. 1989. A Josephson integrated circuit simulator (JSIM) for superconductive electronics application.
- [6] Arash Fayyazi, Shahin Nazarian, and Massoud Pedram. 2020. Logic Verification of Ultra-Deep Pipelined Beyond-CMOS Technologies. *CoRR abs/2005.13735* (2020). [arXiv:2005.13735](https://arxiv.org/abs/2005.13735) <https://arxiv.org/abs/2005.13735>
- [7] Coenrad Fourie. 2018. Single Flux Quantum Circuit Technology and CAD overview. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–6. <https://doi.org/10.1145/3240765.3243498>
- [8] Andrew M. Haslam, Kurt M. English, Alexander Derrickson, and John F. McDonald. 2019. Automated Verification and Optimization of SFQ Superconducting Circuits. *IEEE Access* 7 (2019), 22843–22855. <https://doi.org/10.1109/ACCESS.2019.2899873>
- [9] W.N.N. Hung, Xiaoyu Song, Guowu Yang, Jin Yang, and M. Perkowski. 2006. Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 9 (2006), 1652–1663. <https://doi.org/10.1109/TCAD.2005.858352>
- [10] Ahmed Irfan, Alessandro Cimatti, Alberto Griggio, Marco Roveri, and Roberto Sebastiani. 2016. Verilog2SMV: A tool for word-level verification. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1156–1159.
- [11] Naveen Katam, Alireza Shafaei, and Massoud Pedram. 2017. Design of Complex Rapid Single-Flux-Quantum Cells with Application to Logic Synthesis. In *2017 16th International Superconductive Electronics Conference (ISEC)*. 1–3. <https://doi.org/10.1109/ISEC.2017.8314236>
- [12] Naveen Katam, Soheil Nazar Shahsavani, Ting-Ru Lin, Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram. 2017. Sport lab sfq logic circuit benchmark suite. *Univ. Southern California, Los Angeles, CA, USA, Tech. Rep* (2017).
- [13] Naveen Kumar Katam, Jamil Kawa, and Massoud Pedram. 2019. Challenges and the status of superconducting single flux quantum technology. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1781–1787. <https://doi.org/10.23919/DAT.2019.8747356>
- [14] Naveen Kumar Katam and Massoud Pedram. 2019. Timing Characterization for Static Timing Analysis of Single Flux Quantum Circuits. *IEEE Transactions on Applied Superconductivity* 29, 6 (2019), 1–8. <https://doi.org/10.1109/TASC.2019.2891166>
- [15] Takahiro Kawaguchi, Kazuyoshi Takagi, and Naofumi Takagi. 2015. A Verification Method for Single-Flux-Quantum Circuits Using Delay-Based Time Frame Model. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E98.A (12 2015), 2556–2564. <https://doi.org/10.1587/transfun.E98.A.2556>
- [16] A. Krasniewski. 1993. Logic simulation of RSFQ circuits. *IEEE Transactions on Applied Superconductivity* 3, 1 (1993), 33–38. <https://doi.org/10.1109/77.233410>
- [17] Daniel Kroening and Mitra Purandare. [n.d.]. *EBCM: Homepage*. <http://www.cprover.org>
- [18] K.K. Likharev and V.K. Semenov. 1991. RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Transactions on Applied Superconductivity* 1, 1 (1991), 3–28. <https://doi.org/10.1109/77.80745>
- [19] Shahin Nazarian, Arash Fayyazi, and Massoud Pedram. 2019. qCG: A Low-Power Multi-Domain SFQ Logic Design and Verification Framework. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. 446–449. <https://doi.org/10.1109/ICCD46524.2019.00069>
- [20] Ghasem Pasandi, Alireza Shafaei, and Massoud Pedram. 2018. SFQmap: A Technology Mapping Tool for Single Flux Quantum Logic Circuits. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. <https://doi.org/10.1109/ISCAS.2018.8351603>
- [21] Ramy N. Tadros, Arash Fayyazi, Massoud Pedram, and Peter A. Beeler. 2020. SystemVerilog Modeling of SFQ and AQFP Circuits. *IEEE Transactions on Applied Superconductivity* 30, 2 (2020), 1–13. <https://doi.org/10.1109/TASC.2019.2957196>
- [22] Georgios Tzimpragos et al. 2020. A Computational Temporal Logic for Superconducting Accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (Lausanne, Switzerland) (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 435–448. <https://doi.org/10.1145/3373376.3378517>
- [23] Clifford Wolf. [n.d.]. Yosys Open SYnthesis Suite. <http://www.clifford.at/yosys/>
- [24] Clifford Wolf. 2017. *Formal Verification with SymbiYosys and Yosys-SMTBMC*. <http://www.clifford.at/papers/2017/smtbmc-sby/slides.pdf>
- [25] Alvin D. Wong, Kevin Su, Hang Sun, Arash Fayyazi, Massoud Pedram, and Shahin Nazarian. 2019. VeriSFQ: A Semi-formal Verification Framework and Benchmark for Single Flux Quantum Technology. In *20th International Symposium on Quality Electronic Design (ISQED)*. 224–230. <https://doi.org/10.1109/ISQED.2019.8697701>