



Prototyping a Tool for Automated Discovery of Asynchronous Reversible Superconducting Circuits

Team 302



Abstract

Modern computers are still limited for more complex scientific problems. The main disadvantage of these computers is they are unable to use all the energy provided to them which also makes them run hotter and slower. This disadvantage is referred to as low power efficiency and it also limits the speed of computers.

Our project is based on using a new method to solve the problem of low power efficiency. The technology we are working on is called asynchronous ballistic reversible superconducting computing (ABRS). It works by replacing some of the current computer components with much faster components; however, these new components require very cold temperatures to work. There are also problems with using the ABRS technology and we are focusing on one, finding a way to create components that can work with the technology.

We have created a tool that finds components based on a specified task. The combinations of these components are what makes hard problems possible to solve. First, the tool requires the user to tell it what the component does. Next, it looks at all the possible way to create the component and finds ones that match what the user wanted. Then, the tool gives the user visual results of the component. This project offers a possibility for people to make and build computers with this new technology. It also helps advance the research topic of the technology by letting scientists create more complex computers. The tool can also help companies use the ABRS technology and advance it further. If we are able to utilize the full potential of the ABRS technology, then that can lead to new discoveries of even more technologies and can help solve big problems in the world. This technology can effectively make computers operate at 500 times faster than any existing computer.



Keywords: **ABRS, Superconducting, Reversible**



Disclaimer

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. This document describes objective technical results and analysis. Any subjective views or opinions that might be expressed in this report do not necessarily represent the views of the U.S. Department of Energy or the United States Government.



Acknowledgement

Thank you, Michael P. Frank, for sponsoring our project and advising us on the technical work being done. Without your guidance we would have not been able to complete start the project let alone complete it. We are excited to see what the partnership between Sandia and FAMU-FSU COE will bring in the future.

Thank you to Dr. Jerris Hooker and Dr. Sastry Pamidi for sharing with us their technical knowledge about superconductors and providing us with resources to help immerse ourselves with the topic.



Table of Contents

| | |
|---------------------------------------|-----------|
| Abstract | ii |
| Disclaimer | iv |
| Acknowledgement | v |
| List of Tables | x |
| List of Figures | xi |
| Notation..... | xii |
| Chapter One: EEL4911C | 1 |
| 1.1 Project Scope | 1 |
| Project description: | 1 |
| Key goals: | 1 |
| Markets: | 1 |
| Assumptions:..... | 1 |
| Stake holders: | 1 |
| 1.2 Work Breakdown Structure | 2 |
| 1.3 Customer Needs | 3 |
| Customer Statements: | 3 |
| Interpreted Needs/ Requirements:..... | 4 |
| Constraints: | 4 |
| Team 302 | vi |



| | |
|------------------------------------|----|
| Explanation of Results: | 5 |
| 1.4 Functional Decomposition | 6 |
| Graphics: | 6 |
| Explanation of Results: | 7 |
| 1.5 Target Summary | 9 |
| 1.6 Concept Generation | 10 |
| Tools: | 10 |
| Concepts: | 11 |
| 1.7 Concept Selection | 15 |
| House of Quality: | 15 |
| AHP Matrix | 16 |
| Pugh Matrix | 17 |
| Final Selection | 18 |
| 1.8 Spring Project Plan | 19 |
| Late April - Early May: | 19 |
| Mid - Early April: | 19 |
| March: | 19 |
| February: | 19 |
| January: | 20 |



| | |
|---|----|
| Chapter Two: EEL4914C | 21 |
| 2.1 Project Deliverables | 21 |
| Current State of Project:..... | 21 |
| Project Deliverables:..... | 22 |
| 2.2 Operational Manual | 24 |
| Project Overview: | 24 |
| Component/Module Description:..... | 25 |
| Component Description: | 25 |
| Integration: | 27 |
| Operation: | 31 |
| Troubleshooting: | 31 |
| EXTREMELY IMPORTANT (REFERENCES): | 32 |
| Appendices..... | 33 |
| Appendix A: Code of Conduct | 34 |
| Mission Statement:..... | 34 |
| Roles: | 34 |
| Communication:..... | 35 |
| Team Dynamics: | 36 |
| Weekly and Biweekly Tasks:..... | 37 |



| | |
|--|----|
| Statement of Understanding:..... | 37 |
| Appendix B: Functional Decomposition | 38 |
| Appendix C: Target Catalog | 39 |
| References..... | 40 |



List of Tables

| | |
|---|----|
| Table 1: Needs to requirements table..... | 4 |
| Table 2: Functional Decomposition Cross-reference | 6 |
| Table 3: AHP Matrix | 16 |
| Table 4: Pugh Matrix Iteration #1 | 17 |
| Table 5: Pugh Matrix Iteration #2..... | 18 |
| Table 6: Targets Catalog | 39 |



List of Figures

| | |
|---|----|
| Figure 1: Functional Decomposition Levels | 7 |
| Figure 2: Shoe and Water Cycle Vs. Our Functional Decomposition | 11 |
| Figure 3: House of Quality House of Quality | 15 |
| Figure 4: Functional Decomposition Operational Manual | 25 |
| Figure 5: Functional Decomposition Levels Appendix | 38 |



Notation

| | |
|---------|---|
| ABRS | Asynchronous Ballistic Reversible Superconducting |
| XIC | Circuit Design Tool |
| WRSPICE | Circuit Simulator |
| Python | Programming Language |
| JJ | Josephson Junction |



Chapter One: EEL4911C

1.1 Project Scope

Project description:

Build a software tool to search through the available space of asynchronous ballistic reversible superconducting logic circuits and discover new circuits.

Key goals:

- Understanding the literature available and necessary to asynchronous ballistic reversible superconducting logic circuits.
- Learning how to use the circuit simulation tool.
- Creating an algorithm that can traverse through a space of possible circuits and perhaps find new circuits.

Markets:

The tool is for the field of superconducting reversible logic & researchers, semiconductor market, fabrication sites & laboratories, and education (I.E. Universities).

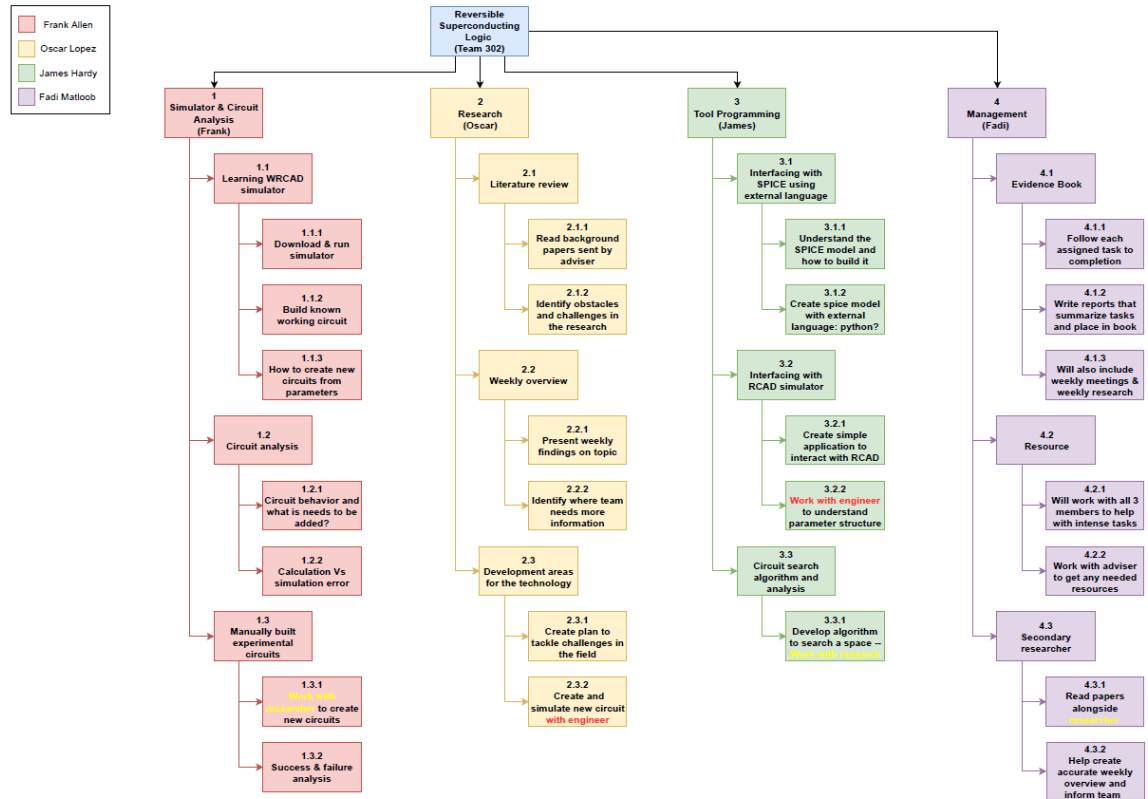
Assumptions:

Simulator can be interfaced with and parameters can be added. Enough information will be gathered to create new circuits.

Stake holders:

Michael Frank, Jerris Hooker, Sandia National Laboratories, FAMU-FSU College of Engineering, Reversible Superconducting Logic research field, and potentially some legal stack holders if any IPs or Patents are developed.

1.2 Work Breakdown Structure





1.3 Customer Needs

Customer Statements:

- Questions & Answers:
 - What is the motivation and purpose of the project?
 - The main motivation is to explore the unknown space of Asynchronous Ballistic Reversible Superconducting Logic circuits. Currently, there is only one working design. The goal, for T302, is to **build a software tool** that attempts to **search through the space of possible circuits** and **acquire new working circuits**. This could pave the way for **learning how to generalize and create design principles** for these types of circuits. Also, the software tool may need to **output circuit parameters that can be taken into a simulator as well as give a graphical representation of the performance of the circuit**.
 - What do you perceive to be the needed roles to effectively build the software tool?
 - Potential roles could include lead programmer and perhaps an experimenter to **test circuits by hand** (Plugging values into a simulator). There is also a **fair amount of research to be done** and knowledge to be accumulated and a role for that maybe considered as well.
 - What language should we use and how should we approach sweeping the parameters?
 - A well supported scripting language would be a good choice like Python. To sweep through the circuits, you have two options. A For-loop based approach where we **sweep the system within a decided space**; however, sweeping time will increase very quickly with the increase in complexity of the circuits. Alternatively, circuits can be **picked randomly until a working circuit is found**. Finally, building the tool to **run in parallel** can be a potential method to help find optimal circuits more efficiently.



Interpreted Needs/ Requirements:

Table 1: Needs to requirements table

| Needs | Requirements |
|---|---|
| 1. Build a software tool to find Asynchronous Ballistic Reversible Superconducting Logic circuits | [1,2] Design an algorithm to sweep the space of possible circuits |
| 2. Reasonable computation time | [1,2] Build the algorithm to have a feasible circuit search time (utilize parallel processing, etc.) |
| 3. Generalize and find circuit design principles | [1,3] Build Interface module to work with the simulator and design tool |
| 4. Obtain background on the topic | [3, 4] Engage in a literature review to understand foundational components of Superconducting Logic circuits (Inductors, Capacitors, Josephson's Junctions, etc.) |

Constraints:

- All circuits will be Asynchronous Ballistic Reversible Superconducting
- WRspice will be used as the simulator



Explanation of Results:

The results given are based on the initial meeting with the adviser/sponsor. We attempted to extrapolate, from the statements given to us, simple and concise needs as can be seen in Table 1. These needs include all the major elements of our project: the circuit space sweeping algorithm, interfacing with preexisting tools (Simulators, etc.), and doing a literature review to understand the topic and create circuit design principles. Also, our software tool needs to be able to find a circuit in a reasonable time (I.E. week or month).



1.4 Functional Decomposition

Graphics:

Table 2: Functional Decomposition Cross-reference

| Subfunctions | Circuit Search | Circuit Test & Display |
|--|----------------|------------------------|
| Enumerate circuit topologies | X | |
| Sweep component values for each topology | X | X |
| Generate WRspice netlist | X | X |
| Run WRspice to simulate operation | | X |
| Interpret and filter failing or non-desired circuits | | X |
| Visualize output by capturing circuit parameters and schematic | | X |

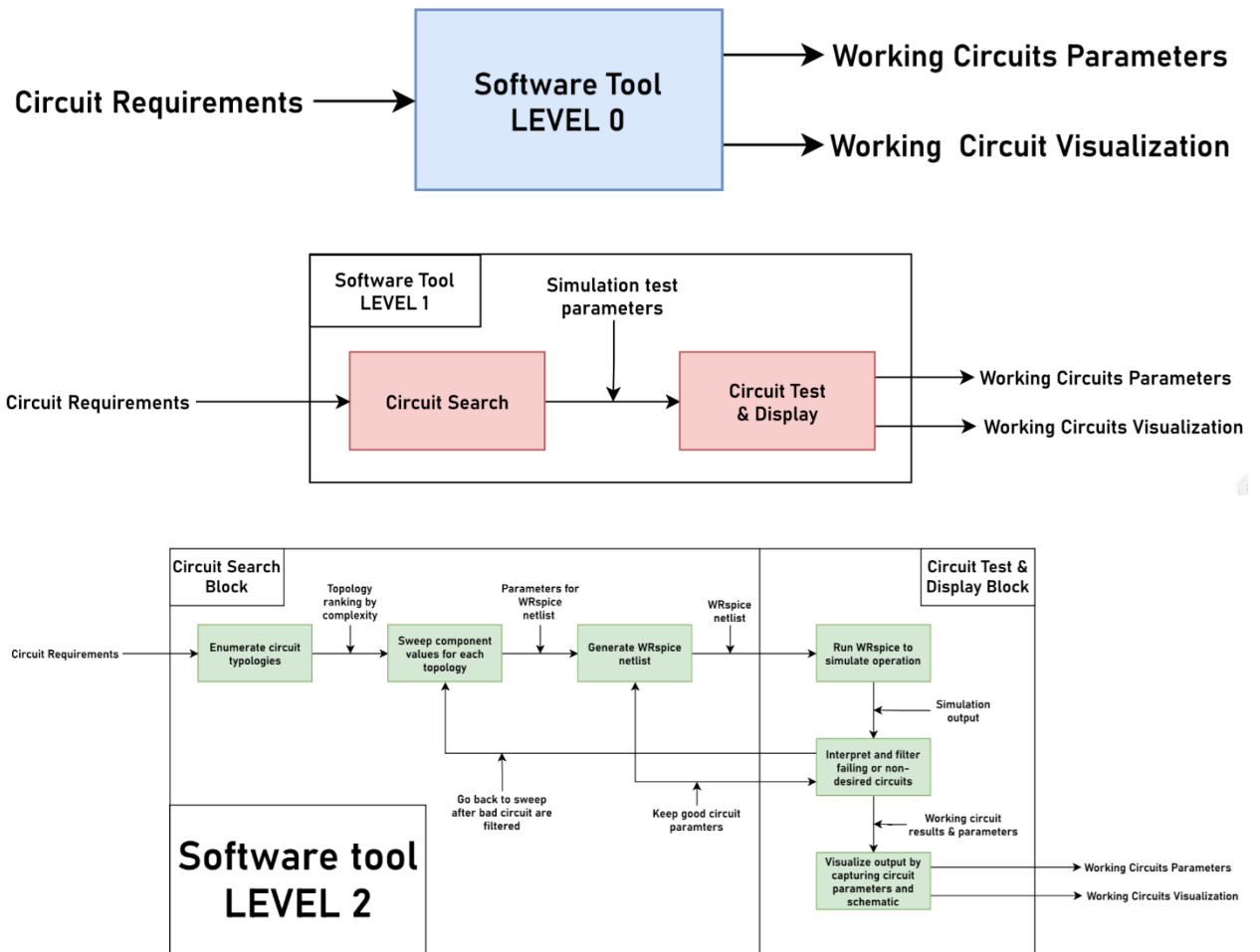


Figure 1: Functional Decomposition Levels

Explanation of Results:

The functional decomposition for this project consisted of three levels with two major functions in level two: Circuit Search and Circuit Test & Display. Then each of the two major functions are broken in subfunctions. The functional decomposition was derived through the resources provided by our sponsor and the conversations we had with our advisor. Table. 1 shows the subfunctions of the software tool as well as the links between the major functions and subfunctions. The table shows that two subfunction modules are utilized in both major functions and therefore there is no need for superfluous copied code. Finally, Figure. 1 shows the functional decomposition flow chart block diagram. The diagram describes the three levels of the



decomposition and clarifies the inputs and outputs of each function as well as their sequence with respect to other functions.



1.5 Target Summary

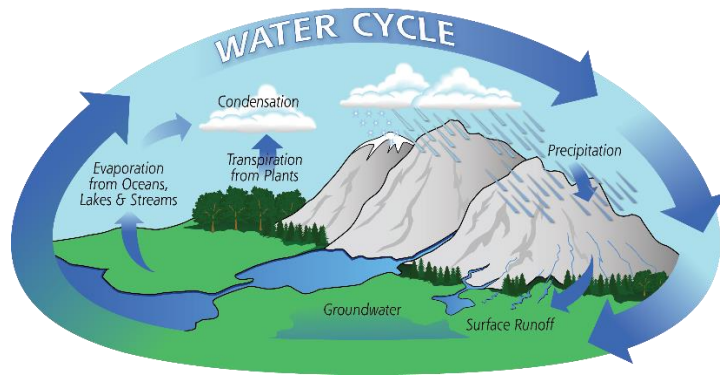
- Generate array of relevant circuit topologies
 - This is an extremely important target as it forms the basis for our project. We chose this target because it accomplishes the main goal of our tool; our goal is to discover new circuits that maybe missed by human designers.
- Sweep through at least 120 in an hour
 - This target is important because it maintains the feasibility of the project. We want to be able to generate a reasonable amount of simulations in a timely manner so that our tool does not take months to turn. The value 120 was chosen by taking 3600 s (1 hr) and dividing by 30 s (average WRSpipe simulation time).
- Generate correct netlist for each circuit
 - This is a very important target as it forms the basis for the circuit simulation. We also chose this target so that we ensure our understanding of superconducting electronics can be utilized to generate circuit parameters in WRSpipe.
- Separate the working from the non-working circuits based on the accepted pre-defined thresholds
 - This is a very important target because we need to be able to classify the different simulated circuits in order to understand their performance. We chose this target based on the intuition that we will have certain thresholds to determine if a circuit is “good enough”.



1.6 Concept Generation

Tools:

- Anti-problem: What would the tool look like if didn't do its job?
 - It would take an incredibly long time to run and get results
 - Very hard to use
 - Requires a lot of user-based optimization
- Force analogy: How does a shoe resemble our program?
 - You have to put on a shoe
 - We need to first give some circuit requirements to our program
 - There are different types of shoes for different situations
 - Our program needs to deal with multiple layers of complexity
- Biomimicry: We compare our program with the water cycle (evaporation, condensation, and precipitation)
 - There is a certain flow to all the processes in our program
 - This helped us generate ideas for the interconnects of the modules



Vs.

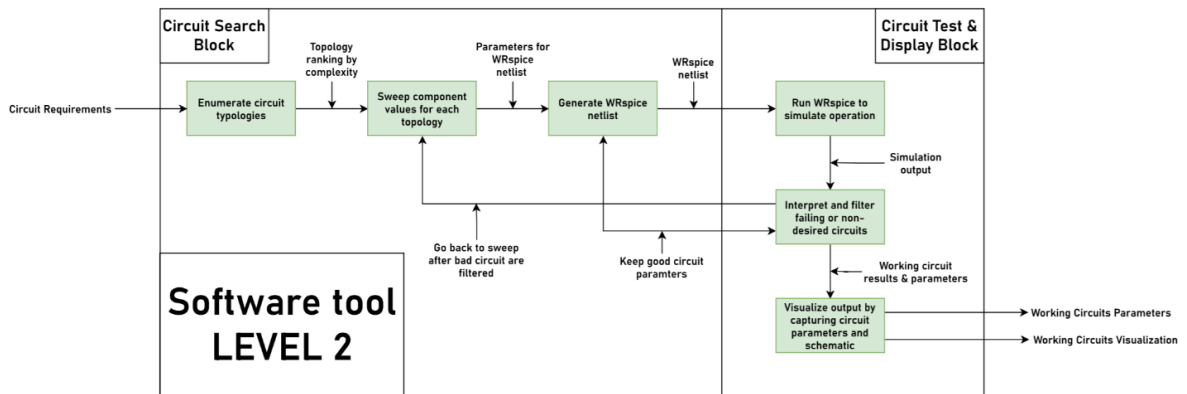


Figure 2: Shoe and Water Cycle Vs. Our Functional Decomposition

Concepts:

Medium Fidelity Concepts:

1. Use Xic to translate netlists to graphical view of circuits
2. Make program translate netlists to graphical view of circuits
3. Generate multiple files based on circuit complexity
4. Team notices correlations between working circuits and changes search parameters on this basis
5. Circuits are found by making it a contest among our peers using a Foldit algorithm



High Fidelity Concepts:

1. Multi-core processing
2. Using premade spice simulator in WRCAD suite
3. Write program in python
4. Use wrspice in command line mode

100 Concepts:

1. Using Windows as main operating system
2. Using Mac as main operating system
3. Using BSD as main operating system
4. Using Unix as main operating system
5. Single core processing
6. Multi-core processing
7. Building our own spice simulator
8. Using premade spice simulator SIMetrix
9. Using premade simulator gnuCAP
10. Using premade spice simulator in Orcad suite
11. Using premade spice simulator Multisim
12. Using premade spice simulator in WRCAD suite
13. Using premade spice simulator LTSpice
14. Using premade spice simulator Ngspice
15. Using premade spice simulator Autodesk EAGLE 8.4 suite
16. Use wrspice in graphical mode
17. Use wrspice in command line mode
18. Write program in C/C++
19. Write program in VHDL
20. Write program in python
21. Write program in Matlab
22. Write program in assembly
23. Write program in Ruby
24. Write program in bash
25. Write program as script instead of traditional program
26. Use Xic to build circuits
27. Use wrspice to write netlists for circuits
28. Use vim to write netlists for circuits
29. Use atom to write netlists for circuits
30. Translate netlists by hand
31. Use Xic to translate netlists to graphical view of circuits
32. Make program translate netlists to graphical view of circuits
33. Find other premade netlists to graphical view of circuits.



34. Create playable game that provides top score for circuit that meets “best working” condition
35. Use matlab to plot results
36. Use wrspice to plot results
37. Generate one large file with all possible circuits
38. Generate multiple files based on circuit complexity
39. Generate one file for each circuit
40. Component values are hard set for each circuit
41. Component values are set as variables
42. Build our own model for Josephson Junctions
43. Utilize level 1 of Josephson Junctions in wrspice
44. Utilize level 2 of Josephson Junctions in wrspice
45. Utilize level 3 of Josephson Junctions in wrspice
46. Utilize verilog to build circuit
47. Use spice to simulate circuits
48. Use pspice to simulate circuits
49. Hire expert on superconductors to search the space we are looking in
50. Become experts on superconductors
51. Use Xyce by Sandia
52. Netlists are written using punch cards then entered into computer
53. Frank changes component values based on max and min range
54. Frank generates netlists
55. Frank systematically changes component values
56. Frank randomly changes component values
57. Frank generates component values into a table in netlist
58. Frank generates random circuits topologies
59. Frank generates systematic circuits topologies
60. Frank focuses generation of circuits topologies based on complexity of circuit
61. Program generates netlists
62. Program systematically changes component values
63. Program randomly changes component values
64. Program changes component values based on max and min range
65. Program generates component values into a table in netlist
66. Program generates systematic circuits topologies
67. Program focuses generation of circuits topologies based on complexity of circuit
68. Program generates random circuits topologies
69. Circuits components are limited to Josephson Junctions
70. Circuits components are limited to Capacitors and Josephson Junctions
71. Circuits components are limited to Inductors and Josephson Junctions
72. Circuits components are limited to Capacitors, Inductors, and Josephson Junctions
73. Circuits components are limited to Capacitors, Inductors, Mutual Inductance, and Josephson Junctions
74. Use darts and dart board to decide which level of complexity to start working on
75. Filter out circuits that we know won't work
76. Filter in circuits that we do know will work



77. Criteria circuit is judged on is power efficiency
78. Criteria circuit is judged on is meeting expected results
79. Criteria circuit is judged on is meeting expected results and power efficiency
80. Test one circuit per day
81. Test multiple circuits per day
82. Program notices correlations between working circuits and changes search parameters on this basis
83. Team notices correlations between working circuits and changes search parameters on this basis
84. Team runs program on centralized server
85. Each team member runs program on personal computers
86. Hire expert on superconductors to teach us about superconductor circuits
87. Hire expert on WRspice to teach us how to use the program
88. Hire expert on Xic to teach us how to use the program
89. Circuits are found by making it a contest among our peers using a Foldit algorithm
90. Circuits are found by making it a contest among general population
91. Program uses genetic algorithm for changing of components values
92. Program uses genetic algorithm for changing of circuit topologies
93. Keep all circuits created and see if they might serve another superconducting purpose
94. Circuits are found by making it a contest among team members, 10 points per working circuit, first to 100 points
bought from available funds
95. Hire software engineer to program front end
96. Apply know circuit rules to see if the hold true in superconductor circuits
97. Utilize a genetic search algorithm to search for circuits
98. Utilize a monte carlo type algorithm to search for the space of circuits
99. Utilize a systematic search algorithm to find new circuits
100. Fabricate circuits at the COE



AHP Matrix

Table 3: AHP Matrix

| Criteria | Weight | HDL | Python | MATLAB |
|------------------------|--------|-----|------------|--------|
| Complexity | 0.2 | 3 | 7 | 6 |
| Interface with WRspice | 0.7 | 0 | 10 | 3 |
| Speed | 0.1 | 7 | 4 | 6 |
| Results | | 1.3 | <u>8.8</u> | 6.9 |

In our AHP matrix we weighted each criterion based on the value provided by our sponsor. Since our sponsor set a constraint for the project, to be able to interface with WRspice, we assumed this was the highest priority. Next, we gave weights to the complexity of each language we used since we are trying to finish the project efficiently. As can be seen, python won because WRspice was made to interact with python which was a huge advantage over the other languages. It is also the simplest language to use.



Pugh Matrix

Iteration #1:

Table 4: Pugh Matrix Iteration #1

| Criteria | Weight | Brute Force | Monte Carlo | Genetic Algorithm |
|---------------|--------|-------------|-------------|-------------------|
| Complexity | 4 | - | -1 | -1 |
| Speed | 2 | - | +1 | +1 |
| Code Required | 3 | - | +1 | +1 |
| Score | | 0 | +1 | +1 |
| Continue? | | No | Yes | Yes |

In this first iteration of the pugh matrix we wanted to decide between different sweeping algorithms. A brute force, systematic, searching algorithm is unrealistic since the complexity of the circuitry is exponential. However, the algorithm is easy to implement. Next, we see that both the monte carlo and genetic algorithm barely edged a win over the reference. So, they needed to be compared against each other.



Iteration#2:

Table 5: Pugh Matrix Iteration #2

| Criteria | Weight | Monte Carlo | Genetic Algorithm |
|---------------|--------|-------------|-------------------|
| Complexity | 4 | - | -1 |
| Speed | 2 | - | -1 |
| Code Required | 3 | - | 0 |
| Score | | 0 | -6 |
| Continue? | | Yes | No |

In this final iteration of the pugh matrix we show the difference between monte carlo and the genetic algorithm. The mote carlo was the winner because it was less complex to implement and was faster than a genetic algorithm implementation.

Final Selection

We did not have many concepts to select from, as most them have already been determined by our sponsor. However, for the choices we had to make, we opted for a software tool that is implemented with python as the language, since it can easily interface with WRspice. We will also utilize a monte carlo algorithm to sweep through the possible circuit space.



1.8 Spring Project Plan

Late April - Early May:

- Graduation
 - Generate circuits with variable complexity
- Engineering design day
 - Results of experiments and circuit generation done
- Finals
 - Documentation and organization to handoff project

Mid - Early April:

- Final Presentation
- Working on poster
- Finalizing test results
 - Creating design principles
 - Documenting methodology
 - Evidence book/manual
 - Verifying results with Michael

March:

- Start testing software tool
 - Generates sensible output
 - Up to 10 complexity *
 - Filter non-working circuits
 - Apply or discover design principles
- Start documenting project handoff to future groups
 - Prepare software tool manual
 - Any materials (papers, posters, etc.)

February:

- Finishing up the tool
 - Combinational circuits (Series + Parallel)
 - Generate combination circuits



- Interface with simulator and visual output
 - Be able to test all generated circuits
 - Can display circuits in XIC electrical view

January:

- Finish parallel and series circuit generation
- Verify results with Michael
- Start building WRSpice interface to insert files and simulate them automatically



Chapter Two: EEL4914C

2.1 Project Deliverables

Current State of Project:

Circuit Generation:

- Series and Parallel circuit generation works perfectly and is optimized.
 - We are able to mitigate any repeating circuit combinations (JJs, Capacitors, Inductors).
- Combinational Circuits work with combining parallel and parallel circuits and the same for series. However, there is no logic to create combinations of parallel and series yet.
 - Once this is done then we can generalize it to multiple combinations.

Sweep Values:

- Currently we have not decided on which algorithms to use; however, we have the ability to implement systematic sweep, logarithmic sweep, monte carlo sweep, and a marginal sweep.
 - Note: the monte carlo sweep and marginal analysis are in the WRSpice the other algorithms would be implemented in python during circuit generation.

WRSpice Simulation:

- Simulations for every netlist can be done in WRSpice.
- Simulation are automated in using python.
 - This is done by calling WRSpice using python and cycling through the different files generated.

Filtering Simulation Data:

- We currently have the ability to test for the existence of a flux across a JJ
- We are able to access data from the simulation and store specific information into a file that can be read using MATLAB and manipulated as data vectors



Visualize Circuit Output:

- We are able to store circuits that we filter to be good and store their parameters and netlists
- There is currently no way of displaying a netlist (This is very very difficult)

Project Deliverables:

Circuit Generation:

- Based on the time we gain access to our files and computer; we would be able to finish combinational circuit generation.
 - This involves any complexity combination of series and parallel circuits combined.

Sweep Values:

- At a minimum, a systematic sweep and a logarithmic sweep will be implemented
 - We will attempt to use the built-in monte carlo sweep and marginal analysis that is provided by WRSpipe

WRSpipe Simulation:

- This section is done.

Filtering Simulation Data:

- We will create a measure for the energy of a fluxon that is produced by analyzing the WRSpipe simulation data and running it through MATLAB
 - This is done by integrating the fluxon SFQ across time.
- We might able to implement a measure for the speed of a fluxon but this more complicated; however, it will be attempted.
 - We take two points and see what the position of the fluxon is and then divide by the time it took for the position to change



Visualize Circuit Output:

- Visualizing circuit schematics will not be done. However, if there is enough time, it will be attempted.
 - Otherwise, this section is also done.



2.2 Operational Manual

Project Overview:

Who is sponsoring the project?

The project is sponsored by Sandia National Laboratories. From Sandia, the direct contact for the project is Michael Frank.

What is this project?

At the highest level, this project is a research project meant to delve into a new research field that is based on a technology known as Asynchronous Ballistic Reversible Superconducting Computing (ABRSC). This technology takes in, as an input, an electromagnetic flux called a Fluxon. The main objective of this project is to create a method to find circuits that can be used to design computing systems (eg. Logic gates). This has to be done as there is yet no formulation of principles to aid building such circuits. In this case, we have opted for creating a software tool that will try different combinations of components to create these circuits and perhaps find something that would be useful for us.

Breakdown of the keywords:

- **Asynchronous:** Fluxons pluses arrive at different time to the device which makes the system a lot simpler than having to synchronize output.
- **Ballistic:** Fluxons self-propagate with very little energy loss.
- **Reversible:** Implies that the state of a device can be recovered. This dramatically lowers power consumption as it rids away with erasing the previous state to have a new one.
- **Superconducting:** This technology utilizes really cold temperatures to maximize power efficiency and obtains frequency at much higher speeds. It also uses a component that is called a Josephson Junction (JJ) that utilizes the Josephson effect (below some critical current, the voltage across the junction is zero). Using a JJ allows for the dissipation of zero

power since $P = I * V$ but V is zero when the current is below the critical current.

Component/Module Description:

What Tools are used?

WRSpice: Free simulator that contains support for the ABRSC components. It uses a file format known as a SPICE netlist that contains the description information of a circuit (eg. Port connection, values, etc.).

XIC: Circuit design tool that allows for schematic build of circuits as well as simulation through WRSpice. There is also the option to create a circuit and generate a netlist file that describes that circuit.

MATLAB: MATLAB is used to manipulate data vectors and data files that come from WRSpice. Mainly used for data analysis.

Component Description:

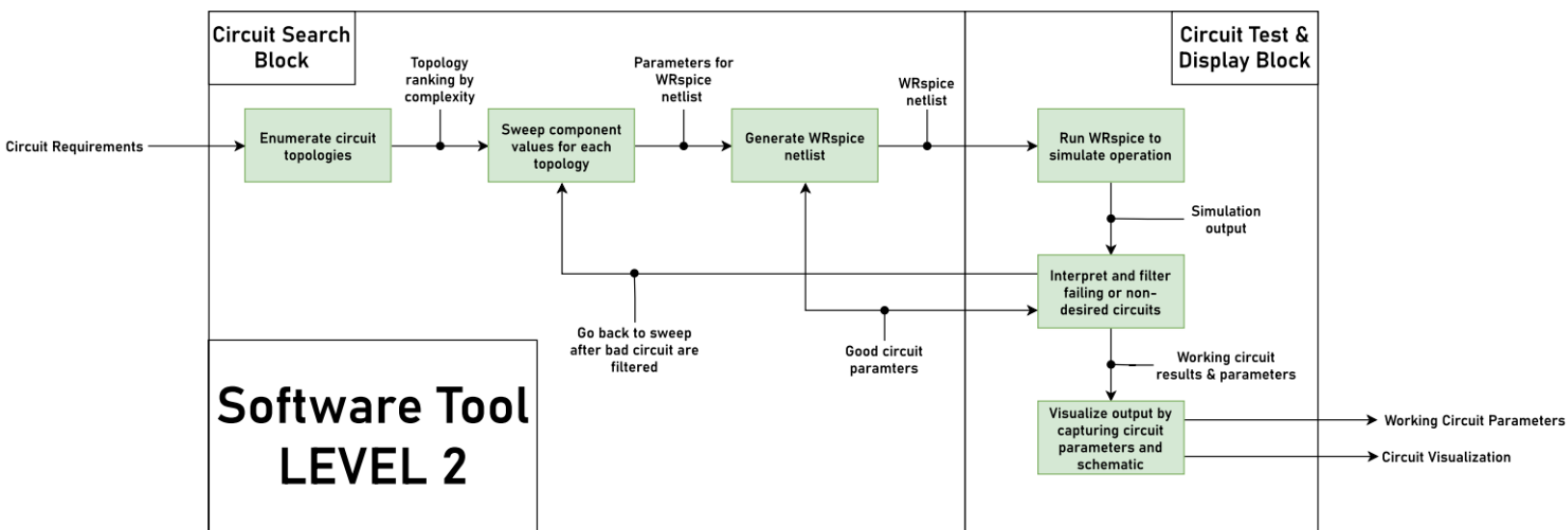


Figure 4: Functional Decomposition Operational Manual

There are 6 main components indicated by green boxes in Fig. 1, these components fall into one of two categories: circuit search & circuit test. The categories are indicated by the boxes in top left and top right corners respectively.



Circuit search/generation:

- **Enumerate Circuit topologies:** There are 3 circuit elements that can be used to create a circuit in ABRSC: capacitor, inductor, and JJ. In the project we also identify a circuit's complexity by the number of components it has. We say that if a circuit had two capacitors with an inductor and a JJ then it would have complexity 4. However, it is important to note that if the capacitors were in series and parallel then they would reduce the complexity to 3 if combined. In this module we implemented a method to abstract the topology generation process to create all possible combinations of N complexity circuits.
- **Sweep component values:** In order to test a topology effectively, we consider changing the values of the components in the topology. We can do this with different algorithms but currently we do a systematic and logarithmic sweep across combinations of values of the components.
- **Generate netlist:** Simply print the node names that the topology created into a file with the syntax of a netlist.

Circuit test:

- **WRSpice simulation:** Automate simulation by calling WRSpice using python and give the netlist files as inputs.
- **Interpret data:** There are metrics for a fluxons energy and existence that can measure the performance of a topology. If the input energy is similar to the output, then that means the circuit was efficient. There are also many circuits where a fluxon may exit the system and that would result in no output; therefore, checking for the existence of a fluxon is rather important.
- **Visualize output:** Ideally there would be a method to display (schematically) a circuit using its netlist; however, this is very difficult. We currently have the ability to store the data from each circuit and output its plots.

Integration:

Since this is a software project the integration is rather simple. Functions call one another to accomplish the tasks that were discussed in the previous section.

The function call graph is shown below:

main → *checkInput*

main → *buildSubcircuits*

main → *pushToWRSpice*

Main first calls **checkInput** which prompts the user to enter their desired parameters for the tool and it also ensures the proper parameters have been passed.

```
def checkInput():
    """Opens file streams based on the library file, main file, and
    testing file directory. ***ERROR CHECKING OF FS***
    INPUT: Filepath Strings OUTPUT: Tuple of filestreams """
    #Input from user
    print("Please enter the relevant information")
    libFile = input("Library file: ")
    mainFile = input("Main Test Circuit file: ")
    testCircuitDir = input("Test Circuit Directory: ")
    #Error checking
    try:
        minComplexity = int(input("Min Complexity: "))
        maxComplexity = int(input("Max Complexity: "))
        libFileFS = open(libFile, 'r')
        mainFileFS = open(mainFile, 'w')
        if not os.path.isdir(testCircuitDir):
            raise Exception(testCircuitDir + " is not a valid directory")
        if not minComplexity > 0 and not maxComplexity >= minComplexity:
            raise Exception("""Min and Max Complexity must be greater than 0 with
                             Max Complexity greater than or equal to Min Complexity""")
    except FileNotFoundError as fnf_error:
        fnf_error = str(fnf_error).replace("[Errno 2] ", "")
        print(fnf_error)
        return None
    except Exception as error:
        print(error)
        return None
    except ValueError or TypeError as val_error:
        print("Both complexity values must be integers")
    else:
        return libFileFS, mainFileFS, testCircuitDir, minComplexity, maxComplexity, False
```

Buildsubcircuits creates a list of components and nodes which these are used to build parallel circuits and series circuits. The logic of the build is found in the source code file.

```
def buildSubcircuits(fileList):
    """Fills the directory of test circuits with viable circuits
    between the minCmplx and maxCmplx
    INPUT:Test Directory FS, lib FS and complexity bounds
    OUTPUT:Full test directory and tracking metric"""
    #Need a metric to track the current circuit and complexity
    validCir = 0
    validPar = False
    validSer = False
    cirList = []
    nodeList = []
    typeArray = set()
    #Open files
    for currComplex in range(fileList[3], fileList[4] + 1):
        serIter = 1
        lastCircuit = False
        nonJJEle = currComplex - 1
        parallelDone = False
        seriesDone = False
        seriesFinal = False
        while not lastCircuit:
            #Generate component List (LOGIC)
            #Complexity 1
            if currComplex == 1:
                nodeList = [Node("aa"), Node("zz")]
                cirList.append(JosJunc(nodeList[0], nodeList[1], 'jjk3'))
                parallelDone = seriesDone = True
            #Parallel
            elif not parallelDone:
                validPar = True
                nodeList = [Node("aa"), Node("zz")]
                if len(cirList) == 0:
                    for num in range(currComplex):
                        cirList.append(Capacitor(nodeList[0], nodeList[1], 1))
                        nodeList[0].addComp(cirList[num])
                        nodeList[1].addComp(cirList[num])
                else:
                    if isinstance(cirList[nonJJEle], Capacitor):
                        nodeList[0].remComp(cirList[nonJJEle])
                        nodeList[1].remComp(cirList[nonJJEle])
                        cirList[nonJJEle] = Inductor(nodeList[0], nodeList[1], 1)
                        nodeList[0].addComp(cirList[nonJJEle])
                        nodeList[1].addComp(cirList[nonJJEle])
                    elif isinstance(cirList[nonJJEle], Inductor):
                        nodeList[0].remComp(cirList[nonJJEle])
                        nodeList[1].remComp(cirList[nonJJEle])
                        cirList[nonJJEle] = JosJunc(nodeList[0], nodeList[1], 'jjk3')
                        nodeList[0].addComp(cirList[nonJJEle])
                        nodeList[1].addComp(cirList[nonJJEle])
                    nonJJEle = nonJJEle - 1
            #Need to check if any parallel components are same type
            for comp in cirList:
                typeArray.add(type(comp))
            if currComplex >= 4 or isinstance(cirList[0], JosJunc):
                parallelDone = True
                cirList.clear()
                continue
```



```

elif not seriesDone:
    validSer = True
    if len(cirList) == 0:
        nodeList = [Node("aa"), Node("zz")]
        #Set up initial full capacitor circuit
        for num in range(currComplex - 1):
            nodeList.insert(num + 1, Node(chr((ord('a') + serIter-96)//26 + 97) + chr((ord('a') + serIter-96)%26 + 96)))
            serIter += 1
        for num in range(currComplex):
            cirList.append(Capacitor(nodeList[num], nodeList[num + 1], 1))
            nodeList[num].addComp(Capacitor(nodeList[num], nodeList[num + 1], 1))
            nodeList[num + 1].addComp(Capacitor(nodeList[num], nodeList[num + 1], 1))
        serIter = currComplex - 1
    else:
        #Change one component ##Need to update node component list
        if isinstance(cirList[serIter], Capacitor):
            cirList[serIter].iNode.remComp(cirList[serIter])
            cirList[serIter].oNode.remComp(cirList[serIter])
            cirList[serIter] = Inductor(cirList[serIter].iNode, cirList[serIter].oNode, 1)
            cirList[serIter].iNode.addComp(cirList[serIter])
            cirList[serIter].oNode.addComp(cirList[serIter])

            for num in range(serIter + 1, currComplex):
                cirList[num].iNode.remComp(cirList[num])
                cirList[num].oNode.remComp(cirList[num])
                cirList[num] = Capacitor(cirList[num].iNode, cirList[num].oNode, 1)
                cirList[num].iNode.addComp(cirList[num])
                cirList[num].oNode.addComp(cirList[num])
            serIter = currComplex - 1
        elif isinstance(cirList[serIter], Inductor):
            cirList[serIter].iNode.remComp(cirList[serIter])
            cirList[serIter].oNode.remComp(cirList[serIter])
            cirList[serIter] = JosJunc(cirList[serIter].iNode, cirList[serIter].oNode, 'jjk3')
            cirList[serIter].iNode.addComp(cirList[serIter])
            cirList[serIter].oNode.addComp(cirList[serIter])
            if len(cirList) > serIter + 1:
                if type(cirList[serIter]) == JosJunc:
                    for num in range(serIter + 1, currComplex):
                        cirList[num].iNode.remComp(cirList[num])
                        cirList[num].oNode.remComp(cirList[num])
                        cirList[num] = Capacitor(cirList[num].iNode, cirList[num].oNode, 1)
                        cirList[num].iNode.addComp(cirList[num])
                        cirList[num].oNode.addComp(cirList[num])
                    serIter = currComplex - 1
                else:
                    serIter -= 0
            else:
                serIter -= 1
        elif isinstance(cirList[serIter], JosJunc):
            serIter -= 1
            continue
        #Generate typeArray
        for comp in cirList:
            typeArray.add(type(comp))
        #Check if this is the last circuit
        if len(typeArray) == 1 and JosJunc in typeArray:
            seriesFinal = True
            seriesDone = True
        #Check if the circuit is valid
        if len(typeArray) == 1 and JosJunc not in typeArray:
            continue

```

```

    """for node in nodeList[1:len(nodeList) - 1]:
        print(node.getName())
        print(node.getCompList())
        print(node.getTypeArray())"""
    for node in nodeList[1:len(nodeList) - 1]:
        if len(set(node.typeArray)) != len(node.typeArray):
            if len(set(node.typeArray)) == 1 and JosJunc not in node.typeArray:
                validSer = False
                break

    """if(len(typeArray) < currComplex and (len(typeArray) == 1 and type(JosJunc) not in typeArray) and currComplex <= 3):
        continue
    else:
        pass"""

    #Combinational
    #Check for compounding
    #Put it in file
    if (not parallelDone and len(typeArray) == len(cirList)) or currComplex == 1 or validSer or seriesFinal:
        with open("{}testfile{}".format(fileList[2],validCir), "w+") as testFS:
            #Always needed
            testFS.write(".lib /home/team302/Documents/wrspice_subckt/main.lib master\n.model jjk3 jj(rtype=0, cct=1, vg=2.8m, icrit=1.5u,
            testFS.write("x0 zz aa master\n")
            for comp in cirList:
                comp.buildCompNet(testFS, cirList.index(comp) + 1)
            validCir += 1
        typeArray.clear()
        #Logical statement that checks all circuits have been created for current complexity
        if parallelDone and seriesDone:
            lastCircuit = True
        cirList.clear()
    return validCir + 1

```

PushToWRSpice calls the WRSpice simulator and passes the netlist files created by the buildSubcircuits function to be simulated.

```

def pushToWRSpice(filepath):
    """Opens a file in WRSpice and the pushes the output to an intermediary file
    INPUT:filepath OUTPUT: output file from WRSPICE"""
    #Save original tty settings and then set it to raw mode(Don't interpret input)
    old_tty = termios.ctgetattr(sys.stdin)
    tty.setraw(sys.stdin.fileno())
    #Open pty to interact with subprocess(master_fd: Python, slave_fd: bash)
    master_fd, slave_fd = pty.openpty()
    #Create a slave process in its own process group
    p = Popen("wrspice -b " + filepath,
              preexec_fn=os.setsid,
              stdin=slave_fd,
              stdout=slave_fd,
              stderr=slave_fd,
              universal_newlines=True)
    #Wait for child process to finish
    while p.poll() is None:
        #Check if stdin or the master
        r,w,e = select.select([sys.stdin,master_fd],[],[])
        #Read from stdin and write to master
        if sys.stdin in r:
            d = os.read(sys.stdin.fileno(), 10240)
            os.write(master_fd,d)
        #Read from master and output to stdout
        if master_fd in r:
            o = os.read(master_fd, 10240)
            if o:
                os.write(sys.stdout.fileno(), o)

```



NOTE: The Sweeping of the values and the filtering of circuits have yet to be implemented but it will simply be an additional function that operate on the topology/circuit that is currently being simulated.

Operation:

Python version: 3.8.6 (others might be compatible)

Running the code:

```
python [NameOfFileToRun] < input.txt
```

OR

```
Python [NameOfFileToRun]
```

Then manually enter parameters.

Parameters needed:

1. Library file: input the path to this parameter. The path is where all the component models are defined as well as the fluxon generator.
2. Main test circuit file: Currently simulated netlist to keep track of current state of the system.
3. Test circuit directory: Where do you want the simulated circuits to go.
4. Minimum complexity: integer for minimum number of components in the topologies
5. Maximum complexity: integer for maximum number of components in the topologies

Troubleshooting:

- Before printing to the netlist file ensure that you are fully understanding what the changes signify. LOGICICAL MISTAKES ARE HARD TO CATCH.
- If WRSpice or XIC are acting out of the ordinary a simple restart usually does the trick.
- Ensure that at least one team member has read the majority of the WRSpice manual to understand how to simulate properly.
- Make sure you have the correct device models installed otherwise the netlists or the simulation may give erroneous results.



EXTREMELY IMPORTANT (REFERENCES):

- **Asynchronous Ballistic Reversible Fluxon Logic** (Michael P. Frank, Member, IEEE, Rupert M. Lewis, Nancy A. Missert, Matthäus A. Wolak, and Michael D. Henry, Senior Member, IEEE)
- **Semi-Automated Design of Functional Elements for a New Approach to Digital Superconducting Electronics** (Michael P. Frank, Rupert M. Lewis, Nancy A. Missert, M. David Henry, and Matthäus A. Wolak Sandia National Laboratories Albuquerque, New Mexico, USA {mpfrank,rmlewi,namisse,mdhenry,mwolak}@sandia.gov)
- **Asynchronous Ballistic Reversible Computing** (Michael P. Frank Center for Computing Research Sandia National Laboratories Albuquerque, New Mexico USA mpfrank@sandia.gov)
- **Schematic for DC to Single Flux Quanta converter (DCSFQ)** - This is what generates the test fluxon <http://www.physics.sunysb.edu/Physics/RSFQ/Lib/AR/dcsfq.html>
- **Applied Superconductivity** – Paul Seidel
- **Dynamics of Josephson Junctions and Circuits** – Konstantin K. Likharev



Appendices



Appendix A: Code of Conduct

Mission Statement:

Team 302 is committed to being accountable and responsible for all tasks and objectives administrated to the team and the individual. Team 302 strives for diligence and efficiency in all tasks and having a mindset of going above and beyond.

Roles:

The roles are described below:

Project Manager: Fadi Matloob

The project manager is responsible for all external communications to the sponsors, advisors, and reviewers. The project manager is only responsible for defining and redefining the project scope as more progress is achieved. The project manager will serve as a bridge between all the roles and will be required to have knowledge in all the areas of the project. Also, the project manager will oversee the delegating tasks to meet deadlines and ensuring the team members perform to the best of their abilities. Finally, the project manager is responsible for delegating “other duties” depending on the task and schedule of team members.

Lead Programmer: James Hardy

The lead programmer will ensure that all tasks containing programming components are completed. The lead programmer will also define the spec for the code to be written so that all the software modules can be integrated together. This includes language interface between the tool and the simulator and vice versa as well as the backend and frontend integration of the application. The lead programmer will have to sometimes work with both the lead researcher and lead engineer to accomplish some tasks



Lead Researcher: Oscar Lopez

The lead research will oversee the literature needed and will point everyone on the team to the resources necessary for each person's tasks. The lead researcher will also be responsible for holding weekly informative meetings to help share findings with the team. Finally, the lead researcher will work closely with both the lead engineer and the lead programmer to help create new circuits and analyze the software tool performance respectively.

Lead Engineer: Frank Allen

The lead engineer will be responsible for understanding how the simulator tool works and should be comfortable in demoing examples to the team. The lead engineer will work with the lead researcher on new circuit designs to understand old models. Also, the lead engineer will help the lead programmer understand the structure of the simulator and help to interface the simulator and the software tool.

Communication:

The main form of communication will be over discord. For team members, Phone and text-messaging will be utilized when discord server is not accessible. For meetings with sponsor, Skype or WebEx will be the primary form of communication. E-mail will be used as the primary non-conference form of communication for issues which are not time sensitive. For the passing of information (i.e. files and presentations), Microsoft OneDrive & Email will be the main form of file transfer and proliferation. GitHub may also be used if needed. Finally, all team members will ensure that their calendars are up to date.



Team Dynamics:

Ethics:

Team 302 is expected to understand and comply with the ethical guidelines provided by IEEE (Institute of Electrical and Electronics Engineers) and NSPE (National Society of Professional Engineers).

Decision Making:

With regards to decisions made for an assigned task, the group member(s) responsible for the task have the final say; however, it is advised and expected that other members' opinions will be taken into consideration but will be require being followed. Project wide decisions will be reserved to the project manager; however, it is expected that the project manager will take into consideration each group members' situation. All decisions made are expected to be beneficial and progressive to the overall project.

Conflict Resolution:

If any issues arise, it is expected that they will be brought up immediately in the next weekly meeting. It is important to keep an open mind with regards to others' perspective and take responsibility for one's own faults. Also, it is expected that conflict will be resolved professionally and not contemptuously. If the conflict needs to be handled immediately, contact all members through discord. Attendance Policy:

All team members must be present at a minimum of 5 minutes before all meetings. If a meeting is going to be missed notification of one day must be given, however a weeks' notice is preferable.

Dress Code:

All professional in-person meetings with the advisor and/or reviewers will be conducted in business casual attire. All weekly meetings within the group or with Dr. Hooker will be conducted in casual attire.

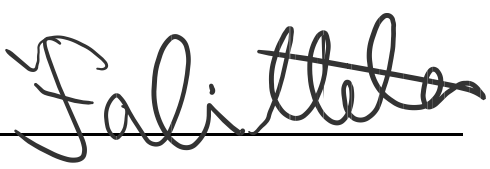

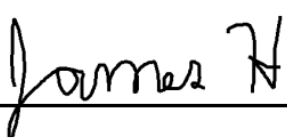



Weekly and Biweekly Tasks:

Group members are expected to report their progress at weekly team meetings that will be held a scheduled time each week. Separately, biweekly meetings will be held with the advisor and short weekly meetings will be held with Dr. Hooker. All progress is expected to be documented in the evidence book.

Statement of Understanding:

By signing this document, the members of Team 302 agree to all the above and will abide by the code of conduct set forth by the group.

| <u>Name</u> | <u>Signature</u> | <u>Date</u> |
|---------------------|---|-------------|
| <i>Fadi Matloob</i> |  | 9/20/2019 |
| <i>Frank Allen</i> |  | 9/20/2019 |
| <i>James Hardy</i> | x  | 9/20/2019 |
| <i>Oscar Lopez</i> |  | 9/20/2019 |

Appendix B: Functional Decomposition

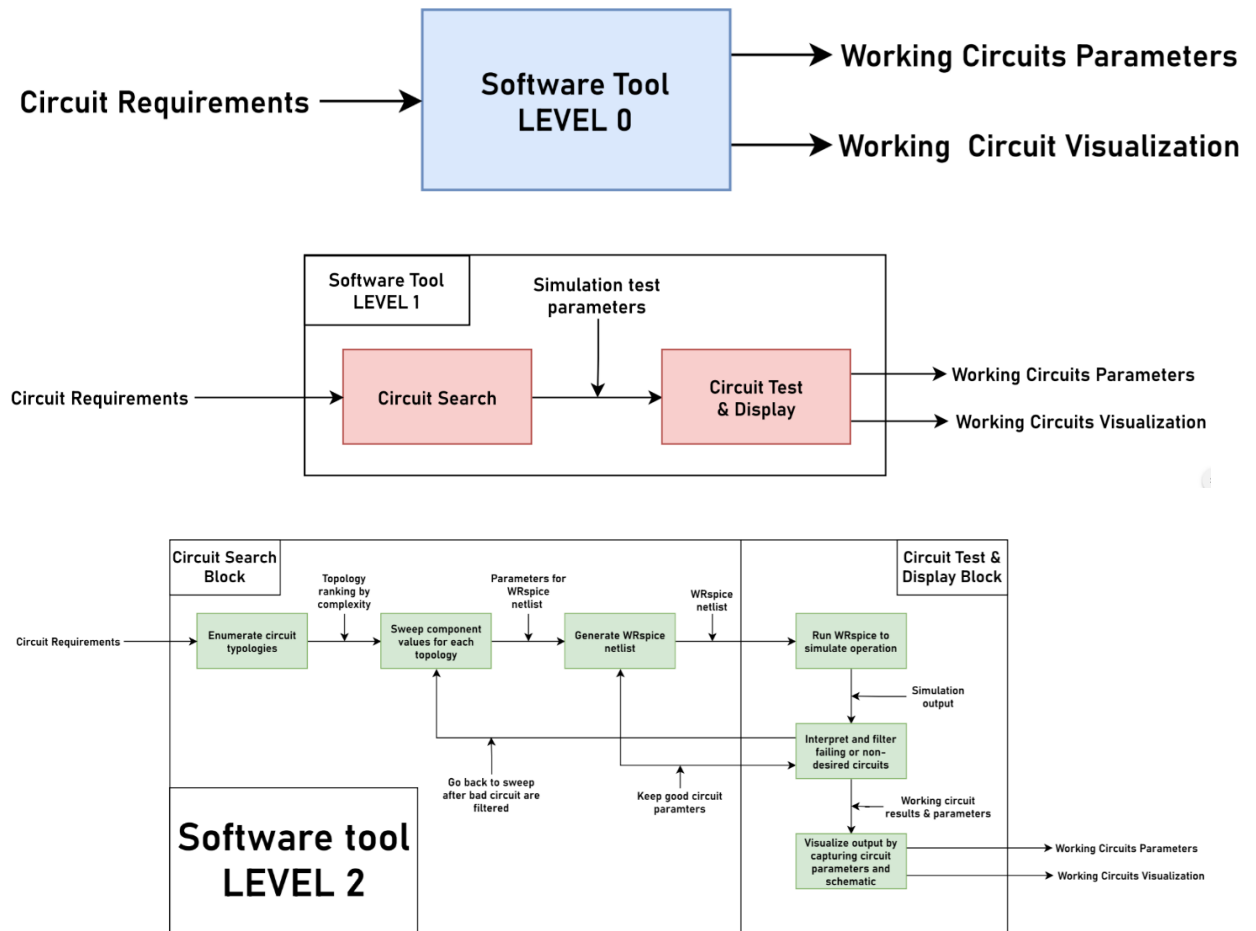


Figure 5: Functional Decomposition Levels Appendix



Appendix C: Target Catalog

Table 6: Targets Catalog

| Requirement | Target | Metric | Units | Ideal Value |
|--------------------------------------|---|--------------------------------------|--------------------------|-------------|
| Enumerate circuit topologies: | Generate array of relevant circuit topologies | Binary | Y/N | Y |
| Sweep parameters | Sweep through at least 120 in an hour | Amount of parameters swept in a hour | parameters/hour | > 120 |
| Generate WRSpice netlist | Generate correct netlist for each circuit | Binary | Y/N | Y |
| Run WRSpice | The simulation produces an output that represents the netlist generated | Binary | Y/N | Y |
| Interpret and filter circuits | Separate the working from the non-working circuits based on the accepted pre-defined thresholds | Binary | Y/N | Y |
| Visualize output | Generate graphical representation of the working circuits in Xic | Binary | Y/N | Y |
| General Targets | Find and generalize design principles governing the ABRC circuits | Binary | Y/N | Y |
| | Contribute to physical circuit fabrication of an ABRC circuit | Binary | Y/N | Y |
| | Optimize circuits to produce maximal power efficient outputs | Measure power efficiency | $\frac{W_{out}}{W_{in}}$ | 1 |



References

- **Asynchronous Ballistic Reversible Fluxon Logic** (Michael P. Frank, Member, IEEE, Rupert M. Lewis, Nancy A. Missert, Matthäus A. Wolak, and Michael D. Henry, Senior Member, IEEE)
- **Semi-Automated Design of Functional Elements for a New Approach to Digital Superconducting Electronics** (Michael P. Frank, Rupert M. Lewis, Nancy A. Missert, M. David Henry, and Matthäus A. Wolak Sandia National Laboratories Albuquerque, New Mexico, USA {mpfrank,rmlewi,namisse,mdhenry,mwolak}@sandia.gov)
- **Asynchronous Ballistic Reversible Computing** (Michael P. Frank Center for Computing Research Sandia National Laboratories Albuquerque, New Mexico USA mpfrank@sandia.gov)
- **Schematic for DC to Single Flux Quanta converter (DCSFQ)** - This is what generates the test fluxon <http://www.physics.sunysb.edu/Physics/RSFQ/Lib/AR/dcsfq.html>
- **Applied Superconductivity** – Paul Seidel
- **Dynamics of Josephson Junctions and Circuits** – Konstantin K. Likharev