GateFinder: um *framework* em Python para encontrar portas lógicas em circuitos de SiDB

Gustavo Guedes de A. Barbosa
Escola de Engenharia Elétrica
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais 31270-901
Email: gustavoguedesab@gmail.com

Arthur Magalhães Fortini
Escola de Engenharia Elétrica
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais 31270-901
Email: arthurmfortini@gmail.com

***Abstract*—Esse artigo apresenta uma versão alfa de um *framework* implementado em Python que utiliza o software SiQAD para auxiliar no processo de desenvolvimento de diferentes portas lógicas, por meio da alteração na disposição de *dangling bonds*. O GateFinder toma como entrada um problema do SiQAD e um programa escrito pelo usuário, gera novas estruturas a partir de alterações na localização dos DBs e analisa quais dessas estruturas ainda se comportam conforme o esperado para a operação desejada.**

I. Introdução

Atomic Silicon Quantum Dot é uma tecnologia que permite o projeto de circuitos lógicos em escala atômica, e foi desenvolvida, inicialmente a partir de observações experimentais [1], explorando conceitos de formação e comportamento de Silicon Dangling Bonds (DBs), que são regidos pelas interações eletrostáticas entre os elétrons. Em linhas gerais, a configuração mais estável de cargas de padrões específicos de DBs permite a criação de portas lógicas e a representação dos clássicos zeros e uns. Já foram implementadas, por exemplo, portas OR [1] e dispositivos de memórias [2] usando DBs.

A exploração dessa tecnologia de maneira manual envolve uma grande quantidade de recursos. Por essa razão, um grupo de pesquisadores desenvolveu uma ferramenta CAD, o SiQAD (Silicon Quantum Atomic Design) [3] para auxiliar no projeto e simulação de circuitos lógicos de ponto quântico, abrindo um novo paradigma de pesquisa para o design de circuitos. Uma vez que a tecnologia supracitada ainda é recente, há muitas regras de design não estabelecidas. A ferramenta CAD torna possível a exploração de tais regras e o desenvolvimento e validação de circuitos lógicos em escala atômica antes de sua produção.

No entanto, o processo de desenvolvimento e análise do comportamento de portas lógicas, por meio da interface gráfica do SiQAD, é lento e dispendioso. O usuário deve gerar um arquivo de design por configuração, definir os parâmetros de simulação, simular e, finalmente, analisar graficamente os resultados. As próximas seções apresentam a estrutura e alguns exemplos de uso do *framework* proposto, cuja finalidade é automatizar algumas etapas desse processo, tornando mais fácil o desenvolvimento e a análise do comportamento de novas configurações de portas lógicas.

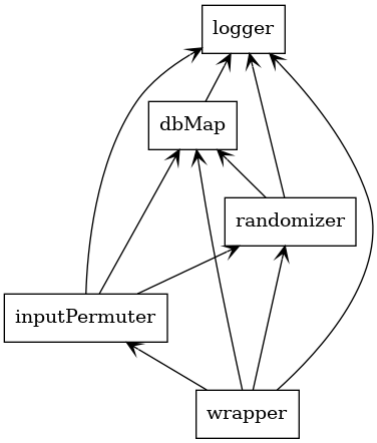

Fig. 1. Relação entre os módulos do GateFinder.

II. Estrutura do Software

A Figura 1 mostra a relação entre os módulos que compõem o *framework*. Os módulos logger, dbMap, randomizer e inputPermuter serão explicados nas subseções a seguir. O módulo com o qual o usuário interage é o wrapper, cujas responsabilidades são instanciar corretamente as classes dos módulos descritos, preparando uma infraestrutura para realizar a permutação das entradas da porta, gerar um diretório denominado *sims*, gerar um design para cada possível permutação da entrada e gerar a tabela verdade para o design, a partir da simulação de cada permutação do valor de entrada.

A. Linguagem e Pacotes Utilizados

A linguagem na qual o *framework* foi escrito é a Python na versão 3.6.9. Os dois principais pacotes utilizados na implementação são o módulo de análise de arquivos xml e o *pysimanneal*, uma interface Python para o *plugin* de simulação SimAnneal. Os arquivos de design, definição de problema e resultado de simulação, utilizados pelo SiQAD, são baseados em xml. Sendo assim, é possível gerar uma abstração da configuração de um design realizando um *parse* dos dois primeiros tipos de arquivos. A partir dessa abstração, um conjunto de operações pode ser aplicado aos DBs e o resultado dessas operações pode ser transformado em um arquivo de design ou definição de problema.

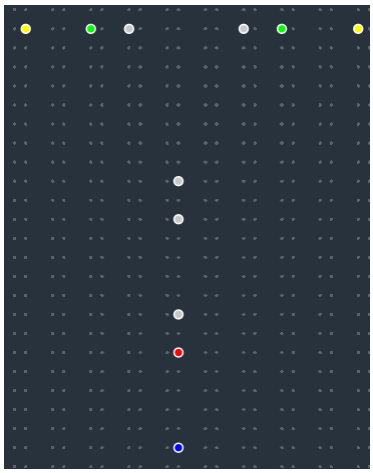

Fig. 2. Porta AND no formato T.

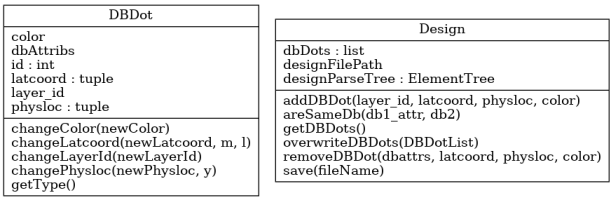

Fig. 3. Classes que permitem a abstração de um *layout*.

B. Classes

1) DBDot: A classe DBDot, pertencente ao módulo dbMap, representa um único SiDB de um design. Ela contém informações sobre a posição, identificador da camada, cor e um identificador próprio. Os métodos dessa classe permitem alterar a cor de um objeto, alterar as informações de posição e identificador de camada e, por fim, a recuperação da informação de tipo. Esse último atributo é uma forma de classificar um DBDot como um valor de entrada, um valor de saída ou um fio de uma porta lógica. Essa classificação foi necessária, pois o SiQAD não fornece uma maneira simples de diferenciar DBs, que não seja suas próprias informações de posição e camada.

Um usuário, ao definir um *layout* inicial, pode especificar quais são os *dangling bonds* que representam entradas e saídas, alterando a cor de cada indivíduo por meio da interface gráfica. Dessa forma, o *framework* é capaz de gerar estímulos nas entradas e avaliar como a saída é influenciada por tais estímulos. A Figura 2 mostra a forma correta de colorizar as entradas e saídas de uma porta lógica para o uso do GateFinder, sendo as cores amarelo e azul utilizadas para representar os perturbadores de entrada e saída, respectivamente, e as cores verde e vermelho, que representam as entradas e as saídas.

2) Design: A classe Design, pertencente ao módulo dbMap, cujo nome é auto explicativo, mantém uma lista de todos os DBs de um arquivo de design ou de problema, além do caminho para esse arquivo e a árvore de análise original. A interface provida por essa classe permite adicionar ou remover *dangling bonds* (addDBDot e removeDBDot), sobrescrever a lista de DBs do objeto (overwriteDBDots), recuperar a lista de DBDots pertencentes a esse design (getDBDots) e, por fim, verificar se dois objetos do tipo DBDot representam o mesmo DB desse design (areSameDb).

3) Permuter: A classe Permuter, pertencente ao módulo inputPermuter, é responsável por separar os DBDots do design por funcionalidade, utilizando a classificação explicada na subseção II-B1 por meio da função mapPorts, e realizar a permutação em si. Essa operação consiste em remover, de maneira ordenada, os perturbadores de entrada, tal que a presença de um perturbador em uma entrada representa o binário 1 e a ausência, o binário 0. Dessa forma, o Permuter procura cobrir todos os possíveis valores de entrada. Ao utilizar a função permute2inputs, as seguintes combinações de dois valores de entrada são analisadas, 00, 01, 10, 11. Já o método permute3inputs possui a mesma funcionalidade, mas para combinações de três valores.

4) Randomizer: A classe Randomizer, pertencente ao módulo randomizer, cujo o nome é contra intuitivo, uma vez que essa classe teve sua funcionalidade alterada em um passado recente, é responsável por movimentar DBs em um design. Uma instância de Randomizer sempre possui um design acoplado, e uma categorização idêntica àquela em II-B3 é realizada durante a construção do objeto. A função dessa classe é alterar o posicionamento dos DBs do *layout* ao qual ela está acoplada.

Há três formas de modificar a disposição de DBs em um design, a primeira é alterar, individualmente, a posição de um *dangling bond*, por meio das funções cujo prefixo é modifySpecificDB. Esses métodos recebem uma forma de identificação de um DB, um valor de distância, que pode ser positivo ou negativo, e um eixo, horizontal ou vertical, formando um vetor de deslocamento. É importante ressaltar que a direção do vetor vertical na ferramenta SiQAD é contra intuitivo, sendo que valores negativos direcionam o movimento para cima.

A segunda forma de alterar a posição de DBs é alterar o ângulo relativo entre um conjunto de *dangling bonds*. Isso é realizado pelos métodos modifyAngle e modifyInputAngle, cujas funcionalidades divergem apenas no número de DBs no conjunto, sendo o primeiro responsável por modificar o ângulo relativo entre dois *dangling bonds* e o segundo responsável por modificar o ângulo relativo entre uma dupla de DBs e um terceiro DB. Essa função é utilizada quando deseja-se alterar o ângulo de um *dangling bond* marcado como entrada e seu respectivo perturbador.

A terceira maneira de modificar a disposição dos *dangling bonds* de um design é alterar a posição da dupla saída e seu perturbador, mantendo a distância relativa entre cada indivíduo dessa tupla. A função modifyPositions toma, assim como as funções modifySpecificDB, um vetor de deslocamento e move a dupla de saída nessa direção.

5) Logger: A classe Logger é responsável por imprimir mensagens de diferentes níveis de depuração. Essa classe é utilizada com a finalidade de facilitar uma definição global do nível de depuração exigido pelo usuário.

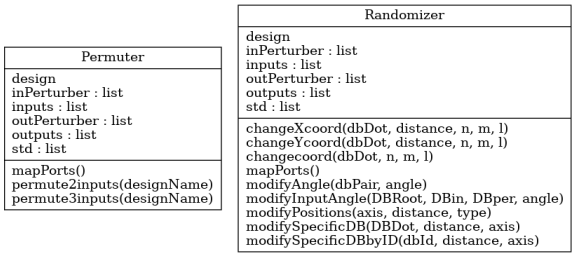

Fig. 4. Classes que promovem a alteração da configuração dos DBs em um design.

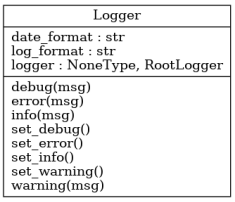

Fig. 5. Classe de geração de registros do *framework*.

III. Fluxos de Uso

Os fluxos de uso do GateFinder são simples e de fácil utilização. Todos tem como peça central o *wrapper.py*, onde devem ser feitas todas as alterações do usuário. A seguir apresenta-se os fluxos *Padrão*, *Apenas Simulação*, e *Design e Simulação sem GUI*. Estes dois últimos fluxos foram derivados do primeiro.

A. Padrão - Descoberta de Novas Portas

O fluxo padrão de uso é aquele para o qual o *framework* foi inicialmente desenvolvido. Nele, o usuário parte de um problema de simulação inicial (ou um arquivo *.sqd* de um design que já tenha sido projetado com a GUI) e realiza modificações no mesmo, utilizando os métodos presentes nas classes do GateFinder. Após isso, todas as combinações possíveis das entradas são geradas e simuladas. Como resultado final, haverá no *cwd* um diretório de nome *sims* contendo subdiretórios, cada um com arquivos *.sqd* e *.xml* com todas as permutações possíveis de entradas. Por fim, também no *cwd* é criado e um arquivo chamado *filename_truth_table.log*, no qual *filename* é definido pelo usuário. Esse arquivo contém um log com a tabela verdade do novo circuito criado a partir das modificações definidas pelo usuário em *wrapper.py*.

Este fluxo foi motivado pois, conforme demonstrado em [4], é possível (e comum) derivarmos novas portas lógicas a partir de outras portas já definidas, que servem como ”base” para modificações.

Em resumo, o passo a passo que o usuário deve seguir neste fluxo é:

1) Criação de um diretório de trabalho, *cwd*
2) Cópia do arquivo-base de *wrapper.py* fornecido na pasta *src* para *cwd*. Em *src* se encontram todas as definições dos módulos descritos na seção II. Dentro do arquivo do *wrapper* há indicações de quais parâmetros devem ser definidos pelo usuário, tais como: quantidade de *inputs*, o *filename* desejado, o valor de μ desejado para a simulação (valores mais comuns são -0.25, -0.28 e -0.32 [3] [4]) e o vetor de potenciais externos, caso o usuário queira simular zonas de clock. Também é indicado em *wrapper* a região na qual as modificações no design devem ser realizadas (por meio dos métodos de *randomizer*). Há também algumas zonas de modificação proibida.
3) Após a conclusão do script, basta que o usuário faça *python3 wrapper.py <design_original>*.

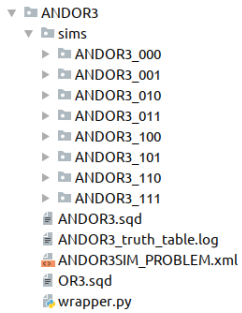

Fig. 6. Hierarquia construída dentro de *cwd* executarmos um fluxo normal de execução do GateFinder.

A Figura 6 mostra um exemplo da hierarquia construída em *cwd* após a execução deste fluxo. Aqui, derivamos uma porta ANDOR de três entradas a partir de uma OR de três entradas. Dentro de *sims/* se encontram vários subdiretórios, um para cada combinação possível das entradas da ANDOR. Na seção IV apresentamos os resultados deste exemplo.

B. Apenas Simulação - Tabela Verdade

Este fluxo é útil para quando o usuário deseja apenas simular algum design já projetado por ele por meio da GUI do SiQAD. Basta que não se faça nenhuma modificação no design original em *wrapper.py*. Dessa forma, todas as combinações de entradas serão geradas e a tabela verdade da porta original será levantada e logada em *filename_truth_table.log*. Vale ressaltar que o primeiro passo do fluxo, nos quais o usuário define nomes e parâmetros em *wrapper.py*, ainda é necessário.

C. Design e Simulação sem GUI

Este fluxo ainda está em desenvolvimento e aperfeiçoamento. Atualmente, por meio dele é possível que o usuário crie um design do zero. Para isso inicialmente é necessário a criação de um design *stub*, contendo apenas um DB em qualquer posição. Então, em *wrapper_guiless.py* o usuário pode utilizar os métodos definidos na classe *(*Design) para adicionar, movimentar e remover DBs em quaisquer coordenadas desejadas. Nota-se que para este fluxo, foi criado um *wrapper* diferente do tradicional. Este arquivo também é fornecido juntamente com os demais arquivos do projeto, em */src*, e nele também estão indicadas as regiões para modificação.

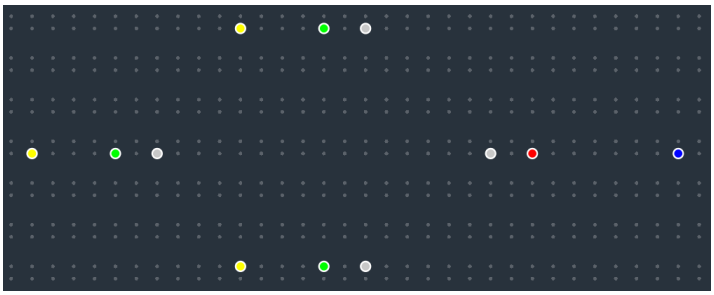

Fig. 7. Porta OR com três entradas.

```
--------------------------------------------------------------------------------
|                              UFMG - 2021                                     |
|                            SiDB GateFinder:                                  |
|                            Truth Table Log                                   |
|                                                                              |
| Design: OR3_NoMod
| Description: SimAnneal results for all possible inputs of                    |
|              modified design.                                                |
| Developed by: Arthur Fortini & Gustavo Guedes                                |
--------------------------------------------------------------------------------

Report start:   2021-03-29 11:05:24.493095

--------------------------------------------------------------------------------
INPUTS                          OUTPUTS
--------------------------------------------------------------------------------
IN0     IN1     IN2
--------------------
0       0       0               0
0       0       1               1
0       1       0               1
0       1       1               1
1       0       0               1
1       0       1               1
1       1       0               1
1       1       1               1
```

Fig. 8. Relatório gerado pelo GateFinder para porta OR de três entradas.

IV. Resultados

A. Apenas Simulação - Tabela Verdade

A primeira e mais simples aplicação do GateFinder é retirar a tabela verdade de um design gerado manualmente pelo usuário. A porta da Figura 7, desenvolvida por uma integrante do laboratório ”nanocomp” do DCC-UFMG (a referência para a produção deve ser inserida aqui quando ela for lançada), foi submetida à categorização manual e, na sequência, à permutação das entradas. O resultado obtido está mostrado na Figura 8.

B. Padrão - Descoberta de Novas Portas

Uma segunda aplicação do GateFinder é realizar alterações nos *dangling bonds* de um design e analisar se essas alterações geram um comportamento que representa alguma lógica útil e mostrar ao usuário qual é esse comportamento. Dessa forma, a partir da porta lógica da Figura 7, foram feitas modificações nos ângulos dos conjuntos de entrada superior e inferior e a porta da figura 9 foi encontrada e se comporta como uma porta ORAND, conforme a equação 1, na qual A, B e C representam as entradas e O a saída. A Figura 10 mostra o relatório gerado pelo GateFinder.

$$O = (A \vee B) \wedge C \tag{1}$$

Além da porta ORAND mencionada, uma segunda porta com comportamento ANDOR também foi derivada a partir do design da Figura 7. A equação da ANDOR está mostrado na

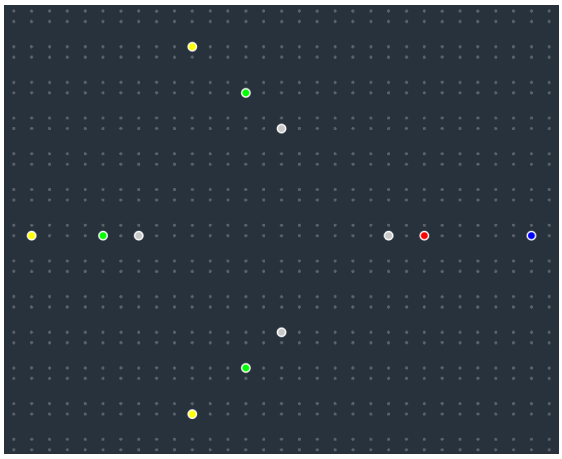

Fig. 9. Porta ORAND gerada a partir da porta OR.

```
--------------------------------------------------------------------------------
|                              UFMG - 2021                                     |
|                            SiDB GateFinder:                                  |
|                            Truth Table Log                                   |
|                                                                              |
| Design: ORAND3
| Description: SimAnneal results for all possible inputs of                    |
|              modified design.                                                |
| Developed by: Arthur Fortini & Gustavo Guedes                                |
--------------------------------------------------------------------------------

Report start:   2021-03-29 13:14:00.501579

--------------------------------------------------------------------------------
INPUTS                          OUTPUTS
--------------------------------------------------------------------------------
IN0     IN1     IN2
--------------------
0       0       0               0
0       0       1               0
0       1       0               0
0       1       1               1
1       0       0               0
1       0       1               1
1       1       0               0
1       1       1               1
```

Fig. 10. Relatório gerado pelo GateFinder para a porta ORAND.

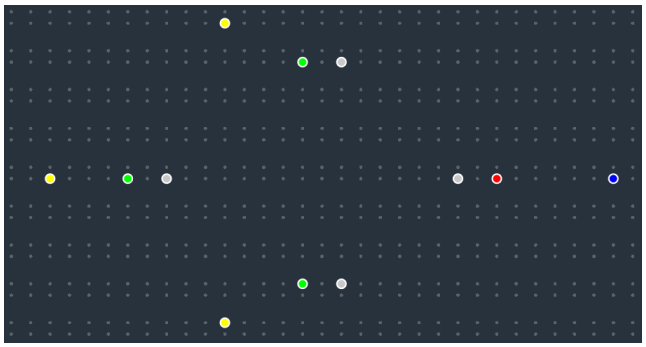

Fig. 11. Porta ANDOR gerada a partir da porta OR.

equação 2, seu design ilustrado na Figura 11 e seu resultado na Figura 12.

$$O = (A \wedge B) \vee C \tag{2}$$

C. Design e Simulação sem GUI

As portas AND e OR do artigo [4] foram geradas utilizando esse fluxo e seus resultados estão mostrados nas figuras 13 e 14, respectivamente.

```
--------------------------------------------------------------------------------
|                              UFMG - 2021                                     |
|                            SiDB GateFinder:                                  |
|                            Truth Table Log                                   |
|                                                                              |
| Design: ANDOR3                                                               |
| Description: SimAnneal results for all possible inputs of                    |
|              modified design.                                                |
| Developed by: Arthur Fortini & Gustavo Guedes                                |
--------------------------------------------------------------------------------

Report start:   2021-03-29 13:45:41.152898

--------------------------------------------------------------------------------
INPUTS                          OUTPUTS
--------------------------------------------------------------------------------
IN0     IN1     IN2
--------------------
0       0       0               0
0       0       1               1
0       1       0               0
0       1       1               1
1       0       0               0
1       0       1               1
1       1       0               1
1       1       1               1
```

Fig. 12. Relatório gerado pelo GateFinder para a porta ANDOR.

```
--------------------------------------------------------------------------------
|                              UFMG - 2021                                     |
|                            SiDB GateFinder:                                  |
|                            Truth Table Log                                   |
|                                                                              |
| Design: AND2                                                                 |
| Description: SimAnneal results for all possible inputs of                    |
|              modified design.                                                |
| Developed by: Arthur Fortini & Gustavo Guedes                                |
--------------------------------------------------------------------------------

Report start:   2021-03-29 15:42:14.043966

--------------------------------------------------------------------------------
INPUTS                          OUTPUTS
--------------------------------------------------------------------------------
IN0     IN1
--------------------
0       0               0
0       1               0
1       0               0
1       1               1
```

Fig. 13. Relatório gerado pelo GateFinder para a porta AND.

```
--------------------------------------------------------------------------------
|                              UFMG - 2021                                     |
|                            SiDB GateFinder:                                  |
|                            Truth Table Log                                   |
|                                                                              |
| Design: OR2                                                                  |
| Description: SimAnneal results for all possible inputs of                    |
|              modified design.                                                |
| Developed by: Arthur Fortini & Gustavo Guedes                                |
--------------------------------------------------------------------------------

Report start:   2021-03-29 15:47:54.627417

--------------------------------------------------------------------------------
INPUTS                          OUTPUTS
--------------------------------------------------------------------------------
IN0     IN1
--------------------
0       0               0
0       1               1
1       0               1
1       1               1
```

Fig. 14. Relatório gerado pelo GateFinder para a porta OR.

V. Limitações de Uso

Devido às considerações em II-B3, o *framework* não é capaz de trabalhar com portas de mais de três entradas e mais de uma saída.

Para se projetar um design usando apenas scripting em Python, o usuário deve primeiro criar um *(*stub).

Atualmente, a utilização dos métodos de *randomizer* dependem bastante de que o usuário conheça bem a indexação dos DBs no design original.

VI. Trabalhos Futuros

Oportunidades para inclusão e melhoria de métodos das classes que compõem o *framework* já foram observadas e estão atualmente em desenvolvimento. Além disso, algumas refatorações já estão programadas. Pretende-se também melhorar o fluxo de design e simulação sem GUI, para que, no futuro, o usuário possa fornecer apenas um arquivo de entrada contendo as informações de posicionamento de DBs, sem necessidade de criação de um *stub*. Um maior número de *inputs* também será suportado no futuro.

VII. Conclusão

O projeto de circuitos baseados em SiDBs segue sendo uma área com muitas regras de design ainda inexploradas. Neste artigo propusemos um *POC* de um software destinado a tornar o processo de design e procura de novas configurações de portas lógicas mais rápido e eficiente, e obtivemos resultados promissores. A primeira versão do GateFinder pode ser encontrada no repositório Git em [5] .

References

[1] T. Huff et al. Binary atomic silicon logic. Nat. Electron., v. 1, n. 12, p. 636–643, Dec. 2018.

[2] R. Achal et al. Lithography for robust and editable atomic-scale silicon devices and memories. Nat. Commun., v. 9, n. 1, p. 1–8, Dec. 2018.

[3] S.S.H. Ng et al. Siqad: A design and simulation tool for atomic silicon quantum dot circuits. IEEE Trans. Nanotechnol., v. 19, n. 1, p. 137–146, 2020.

[4] Bahar, A. N. et al. Atomic silicon quantum dot: A new designing paradigm of an atomiclogic circuit.IEEE Transactions on Nanotechnology, v. 19, p. 807–810, 2020.

[5] GateFinder