Magnetic QCA (MQCA) Design and Testing Tool

A Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Master of Science (Electrical and Computer Engineering)

by

Hui Jin Anderson

August, 2012

APPROVAL SHEET

The thesis is submitted in partial fulfillment of the

requirement for the degree of

Master of Science (Electrical and Computer Engineering)

[illegible]

AUTHOR

This thesis has been read and approved by the examining committee

[illegible]

Thesis advisor

[illegible]

[illegible]

Accepted for the School of Engineering and Applied Science:

Dean, School of Engineering and Applied Science

Date:

Abstract

Quantum-dot cellular automata (QCA) nanotechnology serves as an alternative computational paradigm to CMOS technology. Contrary to the extremely low operating temperature required by QCA, magnetic quantum-dot cellular automata (M-QCA) cells would operate at room temperature.

This thesis describes a project to create a design and simulation engine for magnetic quantum-dot cellular automata (M-QCA) and integrate this simulator into one of the prevailing QCA simulation tools - QCADesigner. As the first stage of an ongoing effort on M-QCA simulators made by HPLP group at University of Virginia, Electrical Engineering department, this project creates the bistable simulator of M-QCA which assumes the cell is a simple two-state system for fast simulation speed.

This simulation engine preserves the extensive set of CAD tools and expands QCADesigner's capabilities as a software product of simulating M-QCA models such as wires, majority gate and more complex circuits. Our vision is to provide the community with a testing and design platform for logical and physical realization of M-QCA devices and circuitry.

Acknowledgements

I would like to acknowledge the guidance and academic support of my advisor Dr. M. Stan, it was his motivation and persistence which made this project’s completion possible. I would also like to thank Medhi Kabir, a Ph.D candidate in our HPLP at University of Virginia, for providing me with wisdom and assistance in the theoretical background of the simulation engine. Also I would like to acknowledge the permission and support from the original inventor of QCADesigner - Dr. Walus and his research group at the University of British Columbia, Canada, who demonstrated admirable dedication to the continuous development of QCA and M-QCA research community.

Table of Contents

List of Figures

List of Symbols and Abbreviations

QCA	Quantum-dot cellular automata
M-QCA	Magnetic quantum-dot cellular automata
RAMA	Reconfigurable Array of Magnetic Automata
$\boldsymbol{E_p}$	Magnetostatic energy, which is the dipole-dipole energy or the interaction energy between the two dipoles
$\boldsymbol{E_k^{i,j}}$	Kink energy, which is the energy cost of cells *i* and *j* having opposite polarization; In another word, the magnitude of E_k determines how much of influence cell *i*'s magnetization has on cell *j*.
PBW	Processing-by-wire
GNU	GNU's Not UNIX
GNOME	GNU Network Object Model Environment
GIMP	GNU Image Manipulation Program
GTK	GIMP Tool Kit

1. Introduction

1.1 Background

The conventional electronic MOSFET technology is charge-based where the charges are used to encode digital information (binary bit 0 and 1). Since charge is a scalar and only has magnitude, the logic levels can only be represented by a difference in the magnitude of charge. For example, more charge stored in a device could signify the logic bit 0 and less charge could signify the logic bit 1, or presence of charge could signify the logic bit 0 and absence could signify the logic bit 1. Switching between logic levels means changing the magnitude of charge in the device, unavoidably it causes current flow which results I^2R Joule dissipation (I is current magnitude and R is the resistance in the path of the current). Modern CMOS technology has relied on the reduction of transistor size to achieve the goal of smaller, faster, and more economical electronic devices for the past four decades. However, at the beginning of the 21st century as the semiconductor industry encountered the "red brick wall" difficulties for further scaling of MOSFET transistors, numerous research efforts have been made in seeking a new logic device either to replace or augment CMOS technology in order to sustain the prediction of Moore's Law. Due to the above factors, spin-based computing paradigms have since gained huge amount of attention in the community.

Compared to charge, spin is a vector quantity that has a fixed magnitude with a variable direction (i.e. polarization). Placing an electron in a magnetic field, it will settles into either parallel and anti-parallel states with respect to the applied field because the two states are the only allowed eigenstates, hence the name of bistable states. These two polarizations can encode the logic bit 0 and 1. Switching between logic levels will not require moving the charge in space and causing a current flow, but merely flipping the spin. The energy dissipated during a spin flip is the energy difference between the two bistable states which can be less than the Landauer-Shannon limit of $K_B Tln(1/p)$ where K_B is Boltzmann's constant, T is temperature in Kevin and *P* is the bit error probability.

Compared to charge-based CMOS technology, QCA and M-QCA are two promising contenders for alternative computing paradigms.

Initially proposed by Doug Tougaw and Craig Lent at Notre Dame, QCA accomplishes computation via coulomb interactions of the electrons confined in quantum dots. A typical QCA cell design contains four quantum dots and two electrons hopping among them (see Figure 1.1).

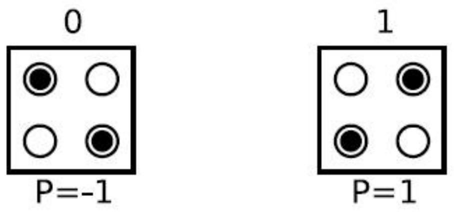


Figure 1.1 QCA logic value definition in terms of QCA cell polarizations (P)

Due to the Coulomb repulsion, the electrons tunnel through the barriers and eventually occupy one of the two diagonal configurations which correspond to the bi-stable ground state representing logic “0” and “1”. With an external perturbation (input), the cell can switch between the ground states in a highly non-linear and abrupt manner. This paradigm is extremely energy efficient, but requires low operation temperature due to the current state of semiconductor technology. This is one fatal limitation of further development and application of QCA computing method.

Magnetic-QCA, on the other hand, is spin-based and utilized the magnetic dipole interactions among the nanomagnet pillars. The magnetization vector of these nanomagnets is analogous to the polarization vector in electronic QCA, but M-QCA model propagates information via magnetic exchange interactions as opposed to the electrostatic in QCA implementations. In a ferromagnetic dot, tens of thousands spins act in unison like a giant classical spin. When the grain has shape anisotropy (magnetic dipolar anisotropy), the magnetization direction becomes bistable, and it can be used to encode logic bits 0 and 1 [1]. This paradigm and its Boolean logic gates have been demonstrated to work at room temperature, that is the immediate advantage of M-QCA over QCA scheme. Cowburn et al. [2] first demonstrated a room temperature M-QCA using a chain of flat ferromagnetic circular dots. Later, the group at Notre Dame developed the M-QCA using non-circular dots and showed the magnetic switching of M-QCA wires with the assistance of an external magnetic field as the clocking field [3]. The clocking fields help to align the magnetization along the hard axis of M-QCA dots and once it is removed, the spins in the dots relax into the ground state set by the initial input without falling into metastable states.

1.2 Motivation

As mentioned above, there exist various versions of M-QCA device modeling, such as ferromagnetic circular dots and non-circular dots. The one created in High Performance Low Power (HPLP) group, in collaboration with department of material science and engineering, at the University of Virginia is based on the simulation needs of a particular version of M-QCA, which is called Reconfigurable Array of Magnetic Automata (RAMA) [4]. It serves as the initial inspiration of our M-QCA simulator. RAMA is designed using arrays of nanopillars. These nanopillars are located at the intersections of a very large and regular cross point structure with wires below the nanopillars running one way, and wires on the top running perpendicular to the wires below. The nanopillars are magnetized, and the selection and geometry of the ferromagnet ensures that the magnetic moments of the nanopillars point either up or down, perpendicular to the substrate. It has already been demonstrated that a random array of up and down polarized ferromagnetic pillars embedded in a ferroelectric or multiferroic matrix can have their magnetizations rotated from being perpendicular to the pillar surface to in-plane direction with the application of a modest electric field [5] as shown in Figure 1.2.

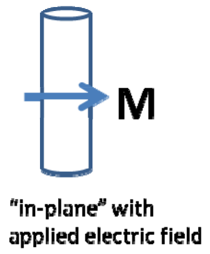


Figure 1.2 Nanopillar magnetic moment rotates to in-plane direction with applied electric field

This version of M-QCA distinguishes itself from the others in the sense that the cell bit features four nanopillars coupled antiferromagnetically and arranged in a square pattern. Each cell bit consists of four pillars in a square with two corners magnetically polarized up and two down due to antiferromagnetic coupling. We discover that this version of M-QCA has a very close structure style to conventional QCA, which has four quantum wells arranged in a square pattern. Thus it would make the most sense that we build and integrate our M-QCA simulation engine into one existing CAD tool and we choose QCADesigner because it is open-sourced free software and is one of the prevailing and mature tool in the field. The M-QCA project aims at creating and integrating the simulation engine into QCADesigner software to complement its functionality and satisfy our needs for designing and testing RAMA and serve other community members that working on the similar M-QCA subjects.

1.3 QCADesigner

QCADesigner is a QCA design and simulation tool developed at the University of Calgary ATIPS laboratory and was originally created by Konrad Walus [6]. The original version of QCADesigner was a simulation engine incorporating the QCA bistable approximation model, and later acquried a GUI using GTK+ as the primary user interface toolkit since it was initially created on Linux(R). QCADesigner provided a platform for testing ideas regarding both the physical realization of QCA devices, as well as the behavior of those devices in QCA circuitry. The original QCADesigner has two simulation engines - QCA Bistable Approximation and Coherence Vector Simulation Engine. The version we used for developing M-QCA has one additional engine - Three State Coherence. All of the engines were developed based on the QCA logic theory.

QCADesigner is an open source software that is written in language C. It employs a wide range of open-source software such as GTK+ graphics library, and is maintained under the GNU’s not Unix public license for open source software. It can be compiled and used on various systems. More software infrastructure background information of QCADesigner is introduced in section 3 of this thesis.

2. M-QCA Bistable Simulation Engine

2.1 Device Idea

The device idea is based on the assumption that we arrange a large and regular array of nanopillars in a square pattern as a cell bit, as illustrated in Figure 2.1 below.

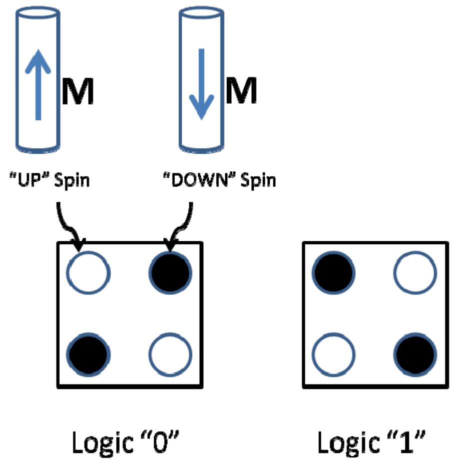


Figure 2.1 Four Pillar Bit Pattern

The ferromagnetic pillars couple antiferromagnetically through their dipolar exchange and thus there are two stable configurations of the array, with each magnet being oppositely magnetized for the two configurations. With an external perturbation (input), the cell can switch adiabatically.

The following subsections briefly describe the theoretical evidences of our algorithms based on which M-QCA bistable simulator was built.

2.2 Micromagnetic Energy System

Landau-Lifshitz magnetic energy equation, E, can be expressed as the sum of the following energy terms: $\boldsymbol{E = E_{ex} + E_D + E_\lambda + E_k + E_H}$ where [7]

- ***E_{ex} is exchange energy***: This is the energy due to the exchange interaction between magnetic dipole molecules in ferromagnetic and antiferromagnetic materials. It is lowest when the dipoles are all pointed in the same direction, so it is responsible for magnetization of magnetic materials. This term can be described as $E_{ex} = -J_{ij}\ \boldsymbol{S_i S_j}$. where J_{ij} is the exchange integral and is a measure of the interaction strength between the two spins. In our M-QCA case, we consider our pillars are bulk magnets, thus this internal material effect can be omitted.
- $\boldsymbol{E_D}$ is **magnetostatic energy**: This is also called dipole-dipole interaction. It represents the field interaction of one dipole onto another. It is dependent on the volume occupied by the magnetic field extending outside the domain. This interaction is most commonly observed in the north/south attraction in magnetic poles. The minimum energy state is created when all north poles are paired with adjacent south poles, that are the total number of unpaired or uncompensated poles is at its minimum.

  Note: This section will be examined extensively in the next section as in our modeling assumption; the other energy terms play insignificant roles in determining the system energy.

- ***E_λ*, E_k** are two sources of **anisotropy energy.** The term anisotropy describes the fact that the properties of a particular magnetic material are dependent on the direction in which they are measured. E_λ is magnetoelastic anisotropy energy and E_k is magnetocrystalline anisotropy energy. This first energy is due to the effect of magnetostriction, a slight change in the dimensions of the crystal when magnetized. This causes elastic strains in the lattice, and the direction of magnetization that minimizes these strain energies will be favored. The second term is the only contribution that is purely intrinsic to the material. The crystal lattice is "easy" to magnetize in one direction, and "hard" to magnetize in others. This energy is minimized when the magnetization is along the "easy" crystal axis, so the magnetization of most of the domains in a crystal grain tend to be in either direction along the "easy" axis. These terms will also be left out for our M-QCA bistable engine as the dynamic process is not of interest and only the final stable states “easy” or “hard” axis are considered. For the next development stage of coherence simulation engine, these terms will assume more significant roles.
- **E_H** is **Zeeman energy**: This is energy which is added to or subtracted from the magnetostatic energy, due to the interaction between the magnetic material and an externally applied magnetic field. The interaction forces the magnetic moment to align with the external field and it can be viewed as the source of causing the nanopillars’ magnetic moments to be “in-plane” direction. When determining the dipole-dipole interactions, we assume no external magnetic field applied, thus this term will also excluded for now.

2.3 Dipole-Dipole Interaction

As mentioned above, in our modeling assumptions, the material related energy terms (i.e. exchange energy and anisotropy energy) will be ignored. When there is no external applied field, Zeeman energy is not a participating factor either; hence the energy that determines the final state of the system is the magnetostatic energy (i.e. the dipole-dipole interaction).

The following paragraphs give readers some background calculation regarding dipole field and energies.

Consider a magnet, or dipole, consisting of two point poles of strength *p*, interpolar distance *l*, and magnetic moment *m* = *pl*. The field H_1 of the magnet at a point *P* distant *r* from the magnet center and in line with the magnet (Figure 2.2.a) is given by, from $H = p/d^2$

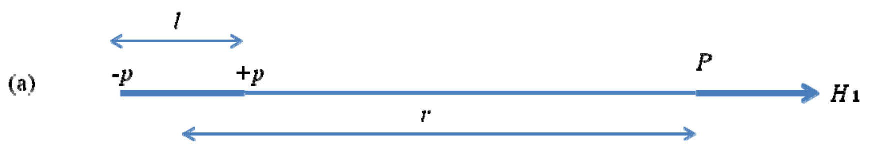
$$H_1 = \frac{p}{[r-\left(\frac{l}{2}\right)]^2} - \frac{p}{[r+\left(\frac{l}{2}\right)]^2} = \frac{2prl}{[r^2-(l^2/4)]^2} \tag{2.1}$$

If *r* is large compared to *l*, the equation 2.1 becomes

$$H_1 = \frac{2pl}{r^3} = \frac{2m}{r^3} \tag{2.2}$$

Similarly, the field H_2 at a point *P* abreast of the magnet center as shown in Figure 2.2.b is the sum of the two fields *H(+)* and *H(-*), equal in magnitude. If *r* is large compared to *l*, we can deduce:

$$H_2 = \frac{pl}{r^3} = \frac{m}{r^3} \tag{2.3}$$



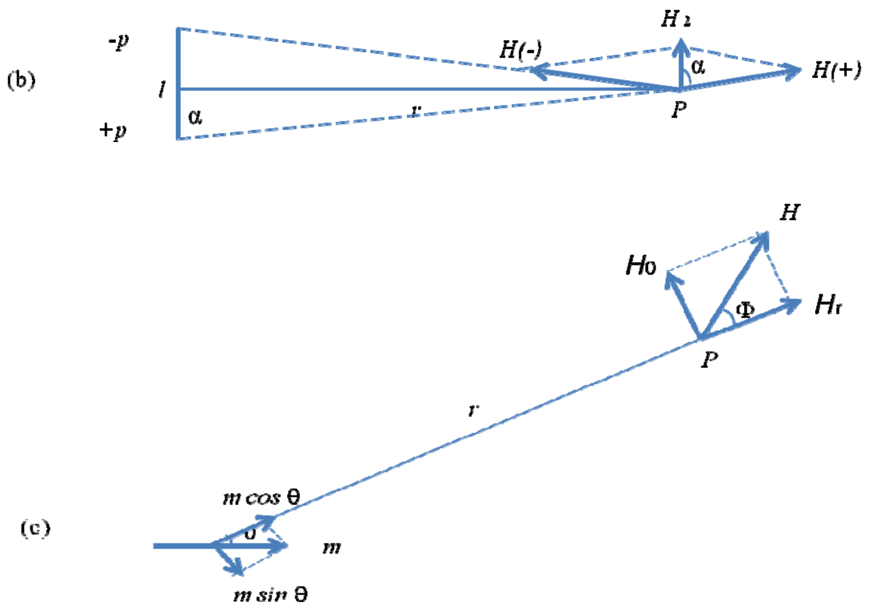


Figure 2.2 Fields of dipoles, adapted from B. D. Cullity *Introduction to Magnetic Materials*

In Figure 2.2.c the line from *P* to the magnet makes an angle *θ* with the magnet axis. The moment of the magnet can be resolved into components parallel and normal to the line to *P*, so that

$$H_r = \frac{2(m \cdot cos\theta)}{r^3} \tag{2.4}$$

$$H_\theta = \frac{m \cdot sin\theta}{r^3} \tag{2.5}$$

A general case of dipole-dipole interaction can be illustrated in Figure 2.3. We want to find out the mutual potential energy of two magnets.

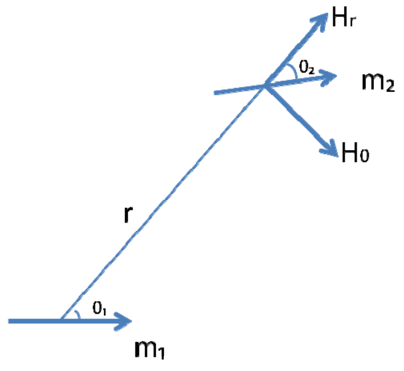


Figure 2.3 Interacting dipoles

This figure illustrates two magnets of moment m_1 and m_2 at a distance r apart and making angles θ_1 and θ_2 with the line joining them. The field at m_2 , parallel to m_2 and due to m_1, is

$$H_p = H_r \cdot cos\theta_2 - H_\theta \cos(90^o - \theta_2) \tag{2.6}$$

We know that potential energy E_p relative to the parallel position *is*

$$E_p = -m.\ H.\ cos\theta, \tag{2.7}$$

hence, combining equations 2.6 and 2.7 we can get

$$E_p = -m_2 \cdot (H_r \cdot cos\theta_2 - H_\theta \sin\theta_2) \tag{2.8}$$

Combining the equation 2.4, 2.5 and equation 2.8, it gives the potential energy of m_2 in the field of m_1

$$E_p = \frac{-m_1 \cdot m_2}{r^3} \cdot (2 \cdot cos\theta_1 \cdot cos\theta_2 - sin\theta_1 \cdot sin\theta_2) \tag{2.9}$$

In another form

$$E_p = \frac{m_1 \cdot m_2}{r^3} \cdot (\cos(\theta_1 - \theta_2) - 3 \cdot cos\theta_1 \cdot cos\theta_2) \tag{2.10}$$

Similarly, it can be shown that the potential energy of m_1 in the field of m_2 is given by the same expression [7], and E_p is the mutual potential energy of the two dipoles, which is the dipole-dipole energy or the interaction energy between the two dipoles, it is fundamentally a magnetostatic energy that we discussed in the section 2.2.

In M-QCA bistable simulation engine, we exam the final states of the system, which means the pillars settle in either parallel or anti-parallel positions. Then the above equation can be simplified to

$$E_p = \frac{m1 \cdot m2}{r^3}, \tag{2.11}$$

when the magnetization of the two pillars are in parallel, where $\theta_1 = \theta_2 = 90^o$, and,

$$E_p = -\frac{m1 \cdot m2}{r^3}, \tag{2.12}$$

when magnetization settles in anti-parallel position, where $\theta_1 = -\theta_2 = 90^o$.

It is apparent that the system would prefer the anti-parallel position, as that is when the overall energy state is at the minimum. E_p here will be used to represent $E_{different}$ and E_{same} which will be used to calculate the kink energy, E_k .

2.4 Kink Energy

Similar to other QCA model in QCADesigner, the kink energy $E_k^{i,j}$ represents the energy cost of cells *i* and *j* having opposite magnetization; In another word, the magnitude of E_k determines how much of influence cell *i*'s magnetization has on cell *j*.

To calculate kink energy between cell *i* and *j*, for each pillar in cell *i* we need to calculate the dipole-dipole interaction between the pillar and each pillar in cell *j*, then sum over all *i* and *j*. Specifically, we need to first calculate the energy when the cells have opposite magnetizations, then when the cells have the same magnetizations, and subtract the two. Mathematically, for neighboring cell *i* and *j*, it is

$$E_{kink}^{i,j} = E_{different}^{i,j} - E_{same}^{i,j} \tag{2.13}$$

The total kink energy experienced by one cell is the sum of all E_k imposed on it by neighboring cells within the effective radius defined by users.

To this end, we can use the total kind energy, the magnetization of neighboring cell j, and the cell-to-cell response function to determine its influence on i, which ultimately leads to the new magnetization of cell i using Ising model [8].

$$M_i = \frac{E_{k}^{i,j} * \left(\sum_j M_j\right)}{\sqrt{1 + E_{k}^{i,j} * \left(\sum_j M_j\right)}} \tag{2.14}$$

The calculation is iterated for all design cells until the new value is converged for a given set of conditions.

3. M-QCA Engine Implementation

The creation and integration of the M-QCA bistable simulation engine was performed in a UNIX-like environment (Linux) using standard open-source tools. The programming language chosen was C. The debug was carried out using GDB, and patch was made using diff utility.

3.1 Platform

The original M-QCA bistable simulation engine was done in matlab and now we want to incorporate it into QCADesigner software to acquire a GUI that can fulfill the following immediate requirements:

1. define a transformation from real cell and pillar coordinates and dimensions to screen coordinates (specified in pixels)
2. place cells precisely on a grid and choose an origin to anchor the real coordinate system for each design
3. equip with a series of CAD application features, such as zoom, selections and pan.

For our M-QCA bistable simulation engine, we need to create ways for users to specify desired parameter values in the relevant options fields, track simulation progress and view status messages, save design work and results, etc.

In order to be comparable to current CMOS Electronic Design Automation (EDA) tools, the software has to meet the above requirements at minimum. The M-QCA bistable simulator has been successfully modeled and compiled into QCADesigner, it allows us to experiment physical and architectural properties of an alternative computation technology via simulation with visual aids before committing the physical implementation, or even explore the future potentials with not-yet existing technologies.

For a complete installation and user guide, please refer to section 5.5 Download and Installation.

3.2 Engine Infrastructure Background

Before performing our simulation engine integration, we need to know the infrastructure involved in creation of QCADesigner and apply appropriate utilization to our M-QCA engine. QCADesigner is built upon several open source libraries and system: Glib, GObject and GTK, which are part of a larger GNU project. GNU project started in 1997 aimed at creating an open source computer desktop environment. Figure 3.1 below demonstrates the relationships among the projects.

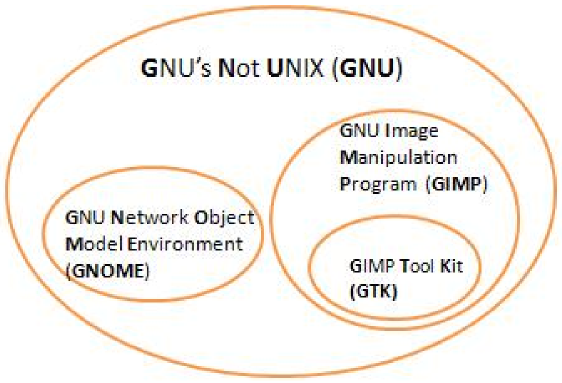

GNU's Not UNIX (GNU)
GNU Network Object Model Environment (GNOME)
GNU Image Manipulation Program (GIMP)
GIMP Tool Kit (GTK)

Figure 3.1 Acronyms and Relationships among the open source projects

3.2.1 Glib

Glib is an acronym of general-purpose library, it provides many kinds of non-graphical implementations that are platform-independent, such as utility functions, data types, and wrapper functions. Glib can be run independently of any other library, however GTK does depend on Glib. Mostly used functions in this software include string handling, memory allocation, file system manipulation, pseudo-random data, process execution, thread management, and many memory data structures.

3.2.2 GObject

GObject is short for the object-oriented runtime type system, and is designed to use directly in language C to provide object-oriented C-based APIs. It typically is distributed in the same package with Glib. It provides functions for declaring new types and for deriving existing types. Each type can be defined as insatiable, classless, fundamental or abstract. Registering a type requires the caller to specify the parent type, as well as how the type's class and instances will be initialized, by providing function pointers to that effect.

GObject is the fundamental type providing the common attributes and methods for all object types in GTK+, Pango and other libraries based on GObject. The *GObject* class provides methods for object construction and destruction, property access methods, and signal support. Most of QCADesinger types are derived from this type. The *GObject* class provides two additional object-oriented subclasses - parameters and signals.

A parameter is a wrapper around one of the data members of the data structure corresponding to instances of declared subclass. Values of the parameter are constrained by the use of *GParamSpec* which declares the parameters' specifications and governs assignment of the resulting parameter.

A signal is a means for an object to notify its environment about a change in its state. Instead of adding application-specific coupling into the object's code, it provides a convenient way to switch the execution context and to ensure that all the involved application reflect the new state of the object. *GObject* signals are declared in the class initialization routine for each subclass and are executed by a signal marshaller. The signal marshaller is a function whose purpose is to call all signal handlers associated with the signal and an instance of the corresponding object class. one particular signal "notify" provided by *GObject* is of our interest and it is widely used in QCADesigner because it can be qualified by the name of the class parameter, which allows all *GObject* parameters to become virtually monitored variables.

3.2.3 GTK

GTK (GIMP toolkit) was originally created for the GNU Image Manipulation Program (GIMP). Licensed under the Lesser General Public License (LGPL), GTK and its advanced versions are adopted as the default graphical toolkit of GNOME and XFCE, two of the most popular Linux desktop environments, providing a library for creating user interface elements (widgets). This toolkit library is also platform independent.

In summary, QCADesigner makes heavy use of GLib's string utility functions, as well as some of the data structures, such as hash tables and doubly linked lists. It is partially object-oriented.

3.3 M-QCA Bistable Engine Source Files

To ensure a successful integration of M-QCA simulation engine into QCADesigner software, we use the same language for coding, and inherent the usage of the libraries mentioned above. Table 1 lists the new source files added to the software that contain the creation of the interface and modeling algorithms of the M-QCA bistable simulator.

Table 1: Newly Created Source Files

File Dir./ Name (./src)	**Description**
MQCA_bistable_properties_dialog.h	Header file of the M-QCA bistable properties dialog
MQCA_bistable_properties_dialog.c	Source file of the M-QCA bistable properties dialog which allows the user to set the parameters for the M-QCA bistable simulation engine
MQCA_bistable_simulation.h	Header file of the M-QCA bistable simulation engine
MQCA_bistable_simulation.c	Source file of the M-QCA bistable simulation engine. This engine treats the system in a time-independent fashion

Table 2 below lists the source files that are modified and updated in order to incorporate the new engine.

Table 2: Modified Source Files

File Dir./ Name (./src)	**Description**
sim_engine_setup_dialog.c	Source file for the simulation engine setup dialog. This dialog allows the user to choose from among the available simulation engines.

Makefile.inc	Makefile include file that contains common set of definition and macros for easy and central maintenance
fileio.h	Header file for the open/save functions
fileio.c	Source file for the open/save functions
simulation.c	It defines the main entry point for the simulation engines, it also contains functions common to all engines
main_batch_sim.c	Source file of a batch mode simulator aimed specifically at verifying outputs against their corresponding inputs
main.c	Source file that initializes GTK, loads saved items, deals with the command line and creates the main window
global_consts.h	Header file that contains all the physical constants and other enumerations
/objects/QCADCell.h	Header file of QCA Cell
/objects/QCADCell.c	Source file for the implementation of the QCA cell.
callbacks.c	Source file that contains the callback functions for main window UI elements

3.3.1 Main Simulation Engine Source Code

MQCA_bistable_simulaiton.h/.c are the main simulation files for physically modeling magnetic QCA designs. The simulation flow follows a series of sequences as illustrated in Figure 3.2:

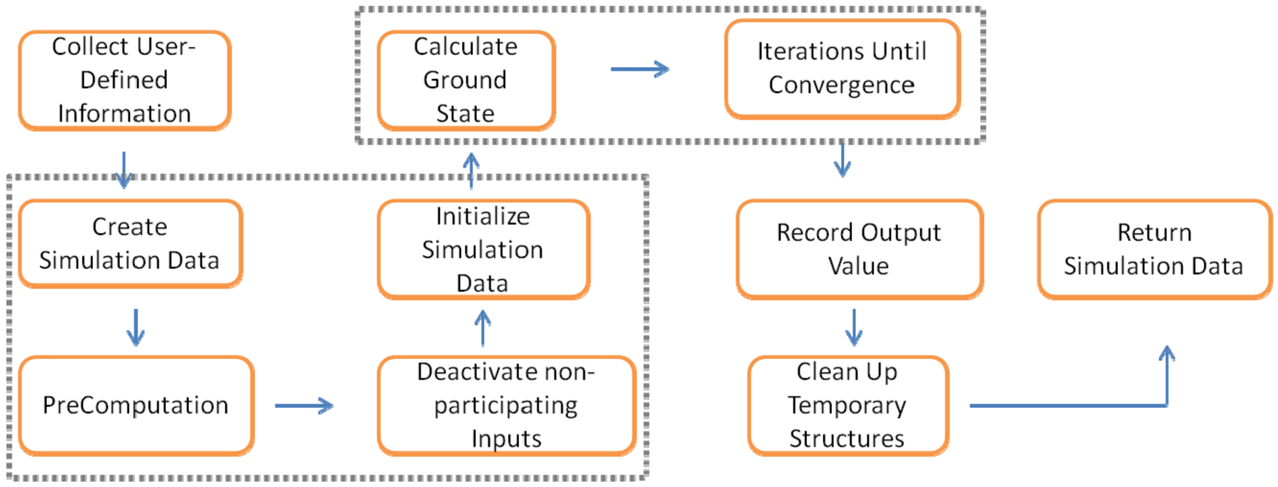

Collect User-Defined Information
Create Simulation Data
PreComputation
Deactivate non-participating Inputs
Initialize Simulation Data
Calculate Ground State
Iterations Until Convergence
Record Output Value
Clean Up Temporary Structures
Return Simulation Data

Figure 3.2 Main Simulation Sequences

This simulation engine receives a design, a vector table, and a structure containing engine-specific options. The engine's first task is to place the parts of the design relevant to the engine into arrays which can be accessed randomly. Data structures are created for each cell to be simulated; these simulation data structure will be filled out with clock values and input values and returned at the conclusion of the simulation. In case the simulation is to be based on a vector table, the user may have chosen to deactivate certain inputs. In such a case, the specified input cells must be "deactivated". This is accomplished by temporarily setting the cell function for the input cells to "normal". Normal cells are not used for either input or output, nor do they have a fixed polarization.

After that, the arrays created during the process as well as the per-cell model-specific data structures are freed using the command g_free. The cell function for inputs marked by the user as “inactive” is reset to “input”, to restore the correctness of the design.

As the last step, simulation engine displays the results of a simulation in a dedicated window, the data waveforms allow users to determine whether the circuit performs correctly.

Note: Contrary to the flat circular cells of the Cowburn Welland experiment, the model we proposed use pillars that has height and volume. So in our bistable model, we need to normalize the magnetization of a pillar to the quantity per unit volume when determining the kink energy among pillars.

The main simulation file defines and uses several data structures. To better understand the codes’ implementation, one may want to familiarize himself with the descriptions of them. Appendix C lists a selection of M-QCA related data structures which are inherited from original QCADesigner.

Two of the newly defined structures in this project are MQCA_bistable_OP and MQCA_bistable_model.

- MQCA_bistable_OP is a structure that contains all the elements for user to specify M-QCA simulation engine options.
- MQCA_bistable_model structure contains the above parameters that characterize the cell and participate in the simulation calculation. This structure is connected to each cell and is unique for M-QCA bistable simulation engine.

3.3.2 Other Modified and Updated Source Code

As indicated in table 2, there are many files updated in order to realize the integration of the engine. A few examples of the important modification are listed below:

- In QCADCell.c define and insert the qcad_cell_calculate_magnetization function and qcad_cell_set_magnetization function in QCADCell.c. However it requires to add cell magnetization property first to QCADCelldot structure. qcad_cell_calculate_magnetization is where the initial magnetism value come from. The idea is that all four pillars are all clocked into the in-plane direction at first, then a bias is applied at the top left corner pillar, which allows the other pillars to evolve through their mutual interactions. Thus we can simplify the calculation and say the magnetization of the cell, which is either 1 or -1, is consistent with the settings of the left corner pillar (the first pillar). Code of the implementation is:

```
double qcad_cell_calculate_magnetization (QCADCell *cell)
 {
 return (cell->cell_dots[0].magnetism);
 }
```

- Create 'MQCA_BISTABLE' instance and apply to all relevant files. To be specific, they are sim_engine_setup_dialog.c, simulation.c, globle_constant.h, mirror_selection_direction_chosen function in callbacks.c, main.c and Makefile.inc.
- In virtually all the listed files in Table 2, include the newly created M-QCA simulation header files, then add the MQCA_bistable_OP structure and *MQCA_bistable_option pointer to roam through the structure.

- Create constants that are used in MQCA_bistable_simulation main function, and include in the global_consts.h, which hosts all the physical constants and other enumerations.
- Other minor changes include that in sim_engin_setup_dialog.h/c files, add "MQCA Bistable Approximation" label and radio box, etc.

Note: one should pay attention to the depth of the #include files. In configuration file, the maximum is set to 5. It may create error messages during compilation if the file level exceeds the number set.

3.4 Compilation and Patch Creation

The bundle of source files and other graphical files are available for download at https://sites.google.com/site/hplplab/resources/software/qcadesigner---mqca-simulation-engine. One may compile and install the software in Linux environment by essentially using the commands below:

- ./autogen.sh
- ./configure
- make
- switching to the src/ subdirectory and run ./QCADesginer

If a user would like to install QCADesinger to a path other than the default, he/she can run configure with the prefix option and specify the path. For example, if you are configuring QCADesigner for the first time, you can run:

- ./autogen.sh --prefix=<Your path>
- ./configure

- make
- make install

In open source projects, the original inventor commonly receive patches or many people publish patches that fix particular problems or add certain functionality. In the communication with QCADesigner original author Dr. K. Walus, we decide to create a patch and make it available to UBC website. However, since a patch file is only useful to whom has the same version of original source codes and QCADesigner software itself has evolved quite a few generations and derived many other versions, the patch file will not be put on HPLP website at UVa as a separate file for download.

3.5 Simulation and Interface

M-QCA simulation engine and other engines in QCADesigner each embodies a different physical model, but performs the simulation tasks in a similar well-defined sequence of steps. They allow circuit designers to examine the behavior of their circuits under given inputs. The following Figure 3.3 illustrates M-QCA simulation operations:

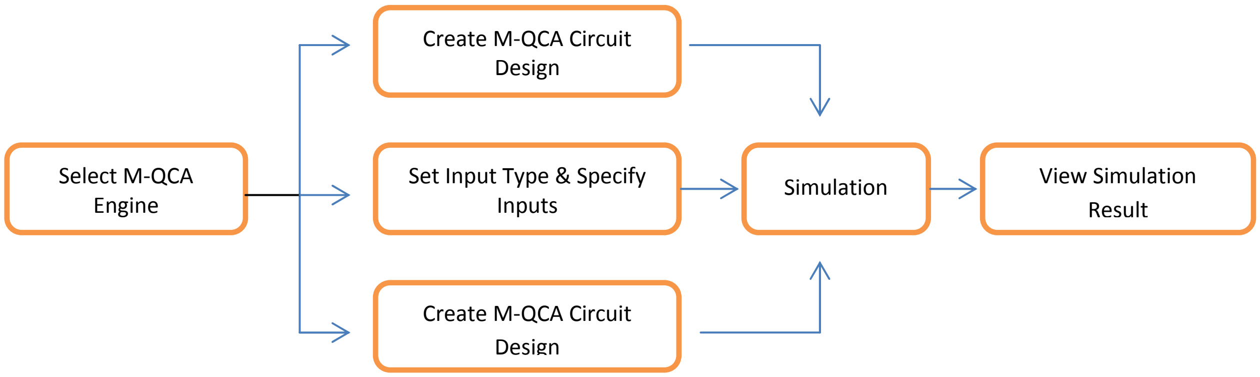
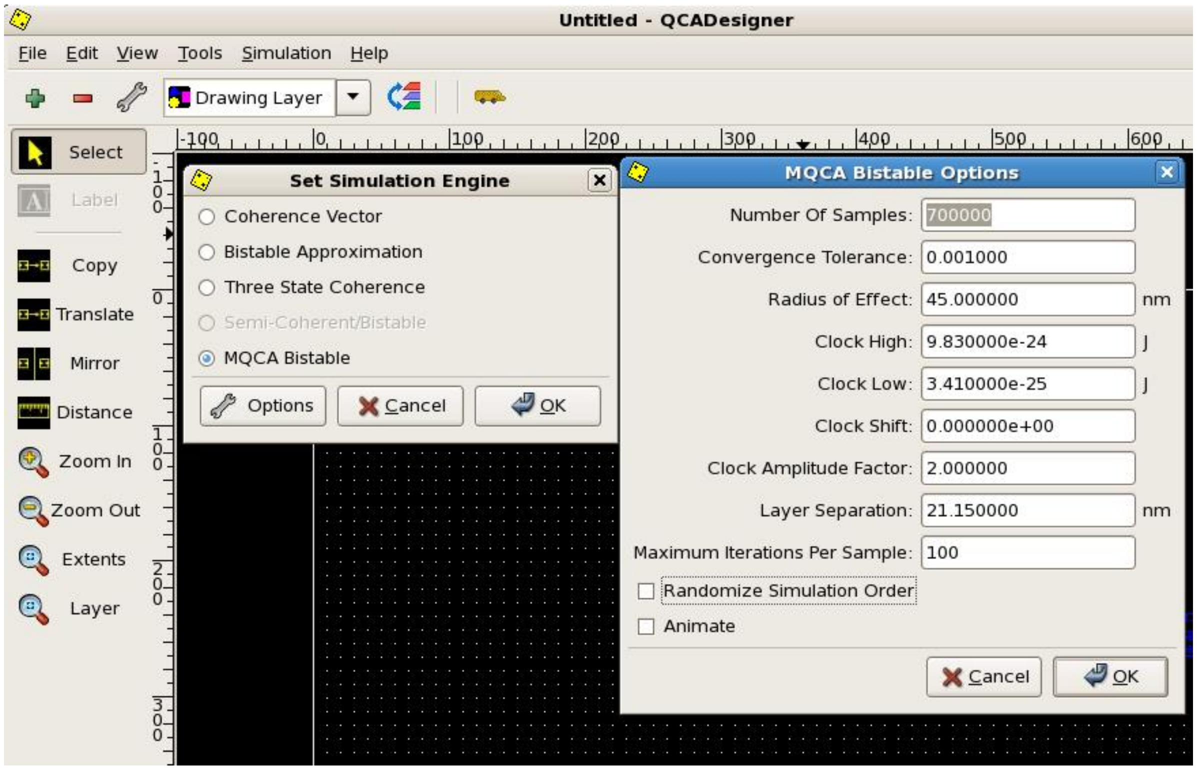

Select M-QCA Engine
Create M-QCA Circuit Design
Set Input Type & Specify Inputs
Create M-QCA Circuit Design
Simulation
View Simulation Result

Figure 3.3 M-QCA bistable simulator interface and options menu.

M-QCA simulation engine main window and dialog boxes have the interface shown in Figure 3.4:



Set Simulation Engine

( ) Coherence Vector
( ) Bistable Approximation
( ) Three State Coherence
( ) Semi-Coherent/Bistable
[✓ MQCA Bistable]

MQCA Bistable Options

Number Of Samples:	700000	
Convergence Tolerance:	0.001000	
Radius of Effect:	45.000000	nm
Clock High:	9.830000e-24	J
Clock Low:	3.410000e-25	J
Clock Shift:	0.000000e+00	
Clock Amplitude Factor:	2.000000	
Layer Separation:	21.150000	nm
Maximum Iterations Per Sample:	100	

☐ Randomize Simulation Order
☐ Animate

Figure 3.4 M-QCA Simulator and Options Interface

For detailed explanations on "Option" dialog box, please refer to Appendix 1: M-QCA Bistable Simulation User Guide.

4. Logic Evaluation

Logic operations are performed by means of the interactions between adjacent cells. This section uses layout graphs to illustrate the popular circuit elements used in more complex designs and demonstrates the corresponding simulation results generated with M-QCA bistable engine.

4.1 Cell-Cell Interaction

The first case to be examined is the simplest scenario with only two cells placed side by side. The first cell is set as the input writing information in the system, and the other cell acts as the output reading the information out after settled in the ground state.

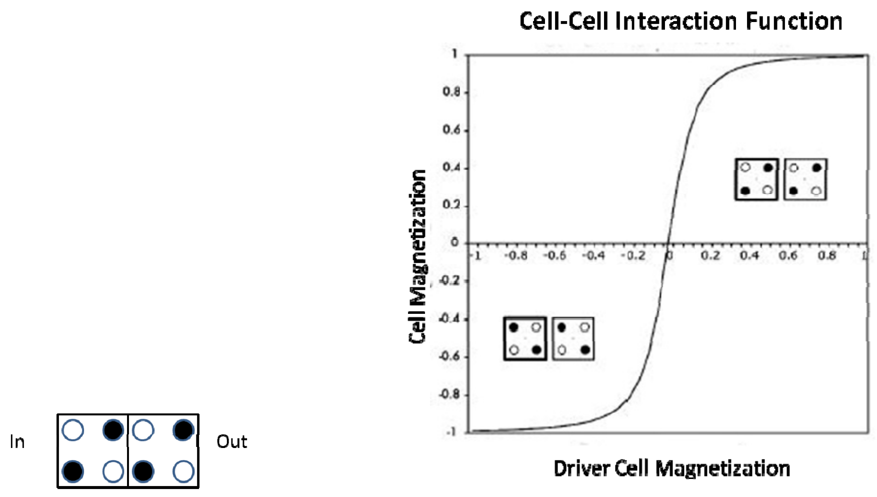


Figure 4.1 Two Cells Analysis. Left: Cell-Cell layout; Right: Cell-Cell Interaction Function

Writing is performed by first clocking the nanopillars by placing them in-plane direction and then applying a small biasing magnetic field in the direction of the written bit. Since neighboring nanopillars are coupled antiferromagnetically, the dipolar exchange field provides the biasing field to write to the appropriate state. Reading of the nanopillar can be done inductively by placing a wire loop over the pillars. When the nanopillar is switched from the in-plane direction to the out-of-plane direction, the change in the magnetic flux in the wire loop induces a current through the loop. The direction of the magnetization determines the polarity of the current through the loop which can be used to read the bit.

As shown in Figure 4.1, there is a nonlinear cell-to-cell response relationship similar to QCA schemes [6].

4.2 Cell Array as Wire

Binary wires with ‘0’ and ‘1’ configurations are demonstrated to work using M-QCA bistable simulator (see Figure 4.2).

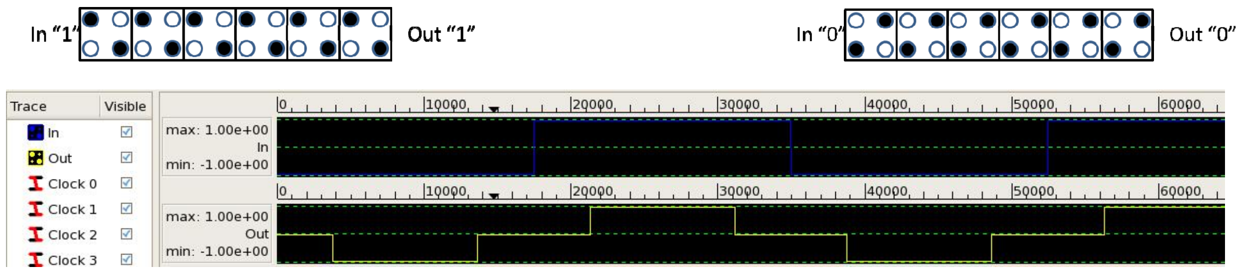


Figure 4.2 Binary Wires. Left: Input “1” configuration; Right: Input “0” configuration; Bottom: Simulation Results using M-QCA Bistable Engine

4.3 Wire Crossover

Wire Crossover is one important design component but also is one ongoing challenge in both QCA and M-QCA circuit design. Traditionally for QCA circuit designs there are three proposals for wire crossover implementations: coplanar crossover, multi-layer crossover and tile-based layout, which in theory should work the same from logic standpoint for our M-QCA case.

Coplanar Crossover: By observing the coplanar crossing, in Figure 4.3, it should be noticed that the vertical wire has all the cells rotated by 45 degrees, relative to the cells in the horizontal wire. In theory, with such organization there is no interference, as two cells with a 45 degree rotation relative to each other have no "kink" energy associated to their polarizations. In other words, for any given non-rotated cell adjacent to a rotated cell, either polarization/magnetization of the non-rotated cell influences the two polarizations/magnetizations of the rotated cell equally, thus eliminates the impacts and maintains its unperturbed state.

But a problem may arise from the increase in distance between the cells of the horizontal wire, right in the crossing point, since the "kink" energy is considerably lower than with the normal spacing, and transmission error may occur.

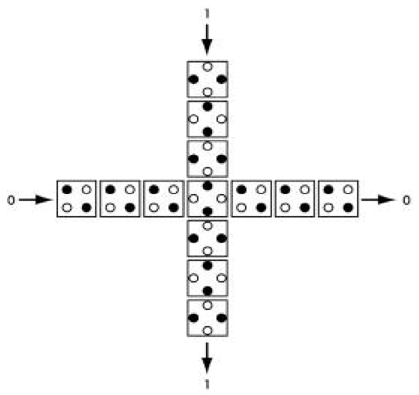


Figure 4.3 Coplanar Wire Crossing.

Multi-layer Crossover: Multi-layer crossover uses the 3D structure, creating additional layers for transmitting signals crossover. See Figure 4.4 for illustration.

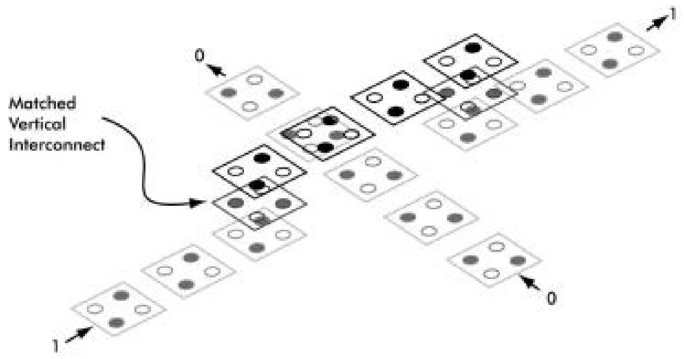


Figure 4.4 Multi-layer Wire Crossing.

Although multi layer crossover gives better simulation results (seems safer), it may not be as easily fabricated as in plane crossovers, thus a compromise between the best (in theory) and the possible must be assumed. And regarding multi layer crossover, it should be noticed that for an even number of separation layers the via connection will act as an inverter.

Tile-based Crossover: This approach takes into account the processing-by-wire (PBW) paradigm of QCA, in which information manipulation can be accomplished while transmission and communication of signals take place. This design scheme creates all logical gates on a 3x3 grids, by populating and depopulating certain cells, various logics are created [9].

The wire crossover is achieved through a gate design show in Figure 4.5 below, which is named “Baseline” gate.

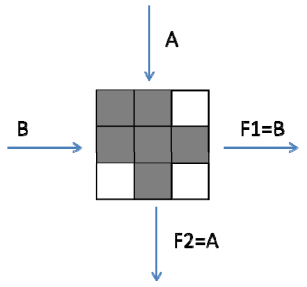


Figure 4.5 Tile-Based Wire Crossing.

Note: From implementation feasibility standpoint, tile-based design has overall advantage, especially for future RAMA circuit. However, for the coplanar and tile-based scheme to work properly, one has to ensure an adiabatic switching of the circuit in the event of changing inputs, so that the system settles gradually in the final global stable state instead of local meta-stable states. Since the magnetization energy levels are higher than QCA cells, we realize that the clocking needs to slow down even more compared to QCA implementation. And since sample number is the variable we can change to vary switching time, contrary to the default sample number of 12800 in QCA engines, we need to set the sample number to approximately 700000 to make the design function properly, which corresponds to a switching time of 0.33ps with the other factors set to the default. More detailed explanations are available in Appendix A.

4.4 Clocking

For M-QCA to work properly and information transfers to the intended direction, we need to apply the minimum three phase clocking scheme, and commonly use four-phase clock signals as shown in Figure 4.6.

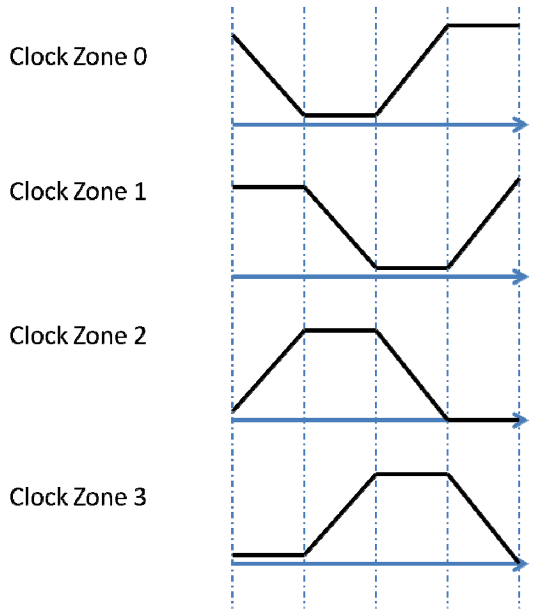


Figure 4.6 Four-phase clocking scheme. Each signal is phase shifted by 90 degrees.

When clock signal is high, the state of the applied cells is forced to NULL, only when the clock signal is low, the subject cells participate in the magnetization calculation in the fashion described above. For an illustration of information flow in a binary wire, please refer to Figure 4.7.

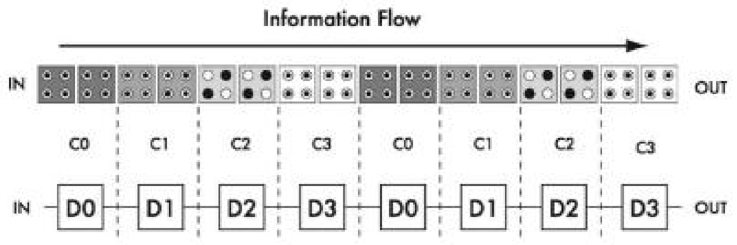


Figure 4.7 Binary wire information transfers to the direction of the applied four-phase clocking.

The clocking field can be implemented utilizing an external magnetic field or in the case of RAMA, a small external electric field. The clocking field helps to align the magnetization along the hard axis of M-QCA dots and once it is removed, the spins in the dots relax into the ground state set by the initial input without falling into meta-stable states.

4.5 Fan-Out

Similarly, a fan-out structure is constructed and verified as shown in Figure 4.8.

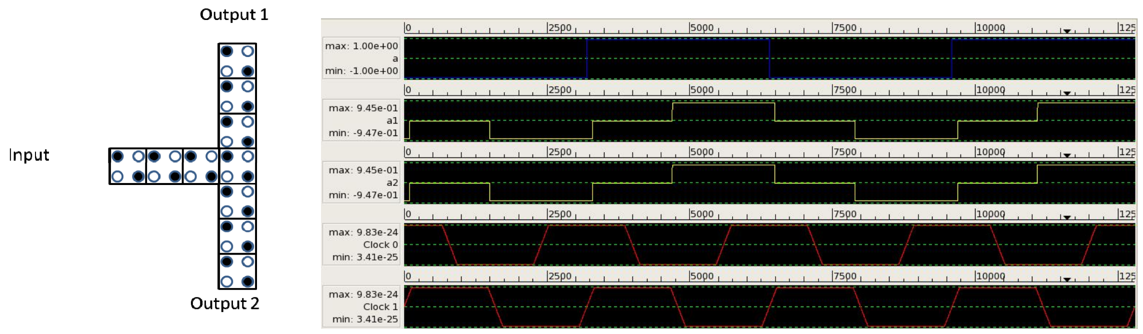


Figure 4.8 Fan-Out layout on the left and simulation results using M-QCA on the right.

4.6 Majority Gate

Majority gate is the fundamental logic primitive available with QCA and M-QCA technology, it is also called majority vote of three inputs which has a structure shown in Figure 4.9.

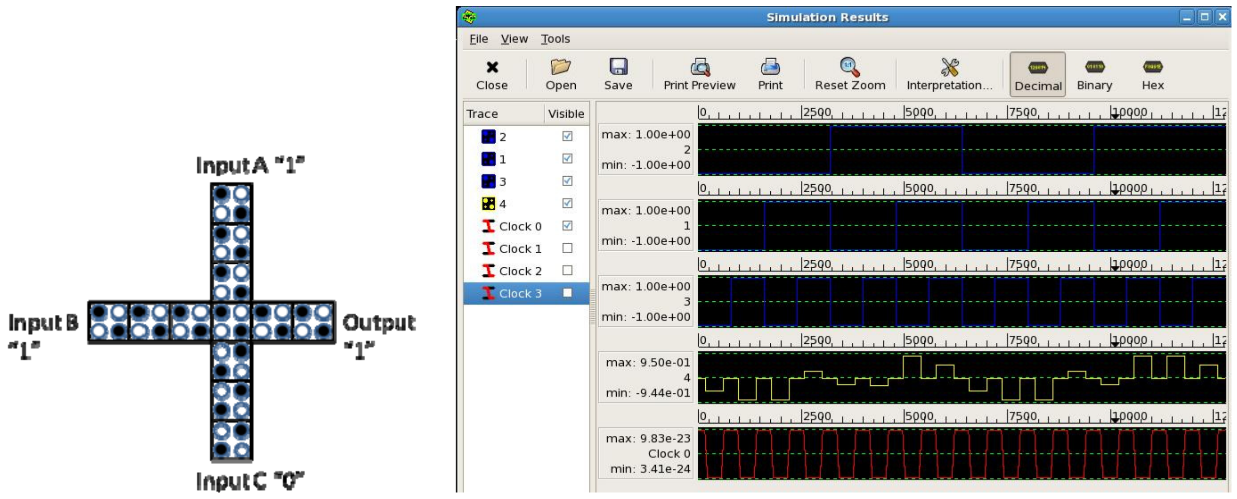


Figure 4.9 Left: majority gate of M-QCA structure; Right: simulation results.

Assuming that the inputs are imposed by cells A, B and C, given the dipole-dipole interactions explained in Section 2.3 and 2.4, and linear superposition of the magnetic field, the central cell will sense the field imposed by the top, left and bottom neighbor cells by assuming a magnetization equal to the magnetization of the majority of these three inputs.

The gate performs the following Boolean function: *Maj(A, B, C) = AB + AC + BC*, therefore the logic product can be performed as *M(a, b, '0')* and the logic sum as *M(a, b, '1').* The truth table for Majority gate is shown in table 3.

Table 3: Logic Truth Table For Three Inputs Majority Gate

A	B	C	MAJ
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	1
1	1	1	1

4.7 Inverter Gate

Inverting function can be achieved through more than one designs. When pillar-pillar positions rotate 45 degrees, it forms an inverter chain, which could yield complement value of the original signal when even numbers of cells are present in the chain. Another more robust design is to form an inverter gate circuit element (see Figure 4.10). Its M-QCA simulation result is placed on the right hand side of the figure.

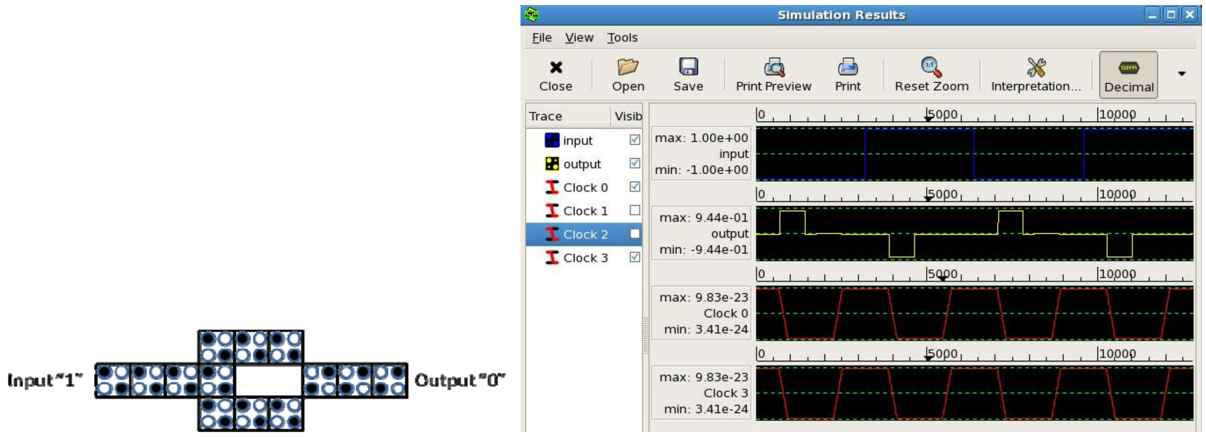

Input "1"
Output "0"
Simulation Results
File View Tools
Close Open Save Print Preview Print Reset Zoom Interpretation... Decimal
Trace Visib
input
output
Clock 0
Clock 1
Clock 2
Clock 3
max: 1.00e+00 input min: -1.00e+00
max: 9.44e-01 output min: -9.44e-01
max: 9.83e-23 Clock 0 min: 3.41e-24
max: 9.83e-23 Clock 3 min: 3.41e-24
0
5000
10000

Figure 4.10 Left: Inverter gate of M-QCA structure; Right: simulation results.

5. M-QCA Circuit Example/Tutorial

5.1 Download

An evaluation version of the tool can be downloaded and the source files locate at https://sites.google.com/site/hplplab/resources/software/qcadesigner---mqca-simulation-engine. Extract the package to a known directory by typing

tar –xzvf QCADesigner-MQCA.1.0.1-src.tar.gz (for tar.gz sources)

tar –xjvf QCADesigner-MQCA.1.0.1-src.tar.bz2 (for tar.bz2 sources)

It can be compiled and installed in Linux systems by following the instruction given in section 3.4 *Compilation and Patch Creation* in the thesis.

5.2 1-Bit Full Adder

In this thesis, a classical example of 1-bit full adder is used to test M-QCA engine here using the elements available to us, such as fan-out, inverter, majority gate and wire crossover etc. A 1-bit QCA full adder was first proposed by P. D. Tougaw and his co-workers in 1994. Since then, the QCA full adder design has been improved. A 1-bit QCA full adder in [10] reduces the hardware count by two majority gates and one inverter. Nowadays, the widely used schematic diagram of the single-bit full adder is shown below in Figure 5.1.

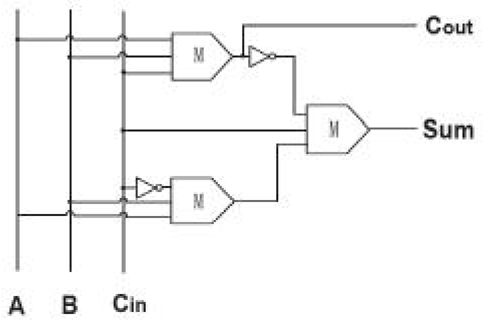


Figure 5.1 1-bit full adder gate based design.

The expressions for the sum and carry output for this adder are: C_{out} = *Maj (A, B,* C_{in}*)* and *Sum = Maj(* C'_{out} *,* C_{in} *, Maj(A, B,* C'_{in} *))*, where the inputs are *A, B,* C_{in} and the outputs are *Sum* and C_{out}. AITPS lab has a step-by-step tutorial on designing a full adder using coplanar crossing [11].

The layout is re-created and tested against M-QCA bistable simulation engine as shown in Figure 5.2. This version of the full adder design is simple and physically easier to implement, however, the error rate associated with coplanar crossover is relatively high as mentioned in Section 4.3.

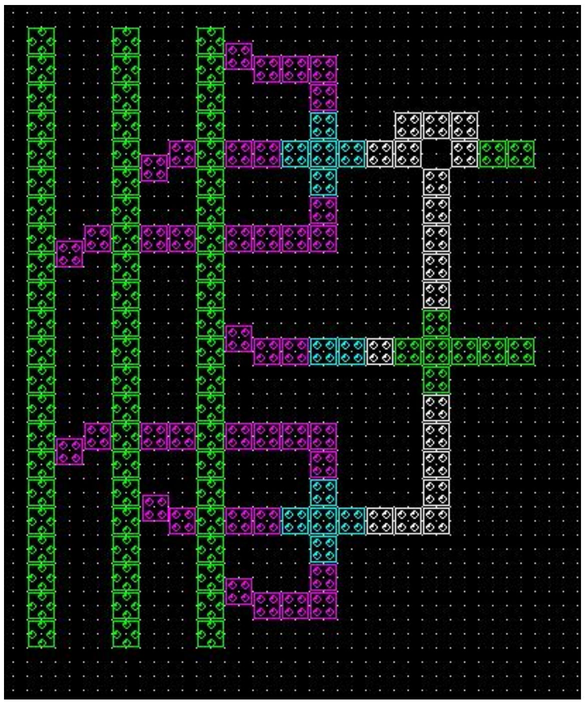


Figure 5.2 1-bit full adder using coplanar crossing method.

Another layout of single-bit full adder using multi-layer crossover scheme was also created and tested in this thesis (see Figure 5.3). Since the reliability is robust relative to coplanar crossover based design, we can decrease the clock switching time by the order of one while still get the correct logical results. As a result, the total simulation time is relatively shorter.

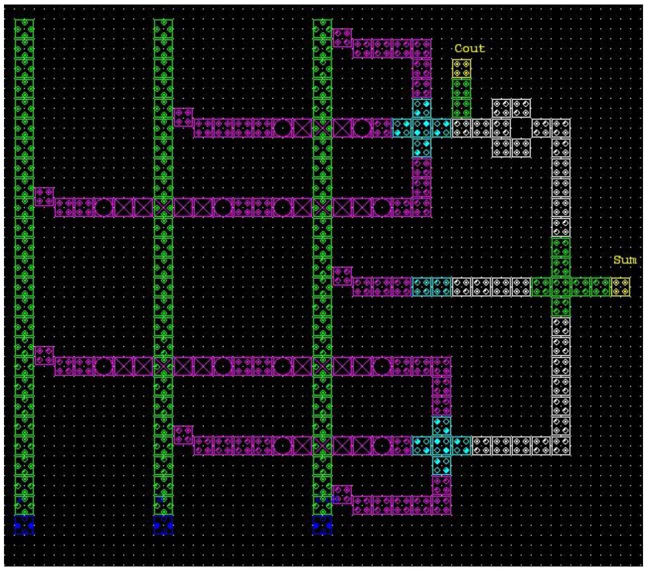


Figure 5.3 1-bit full adder using Multi-layer crossing method.

In summary, when creating the physical layout, the steps and utilized facilities include:

- Drawing QCA cells individually or in arrays, optionally aligned to a grid with a default spacing (20 nm) equal to the default cell size (18 nm) plus the default inter cell spacing (2 nm).
- Setting clock signal for each QCA cell, which is required to have synchronous circuits working properly.
- Drawing QCA cells with 90 degrees rotation, which is required to have Coplanar signal crossing.
- Multi-layer QCA layout design, which is required to have multi layer signal crossing.
- Grouping the input/output signals in buses, to simplify signal name handling, simulation input vectors definition, and simulation results inspection.

5.3 Simulation Results

Simulation results are shown in Figure 5.4 with "vector table" option turned on and utilized.

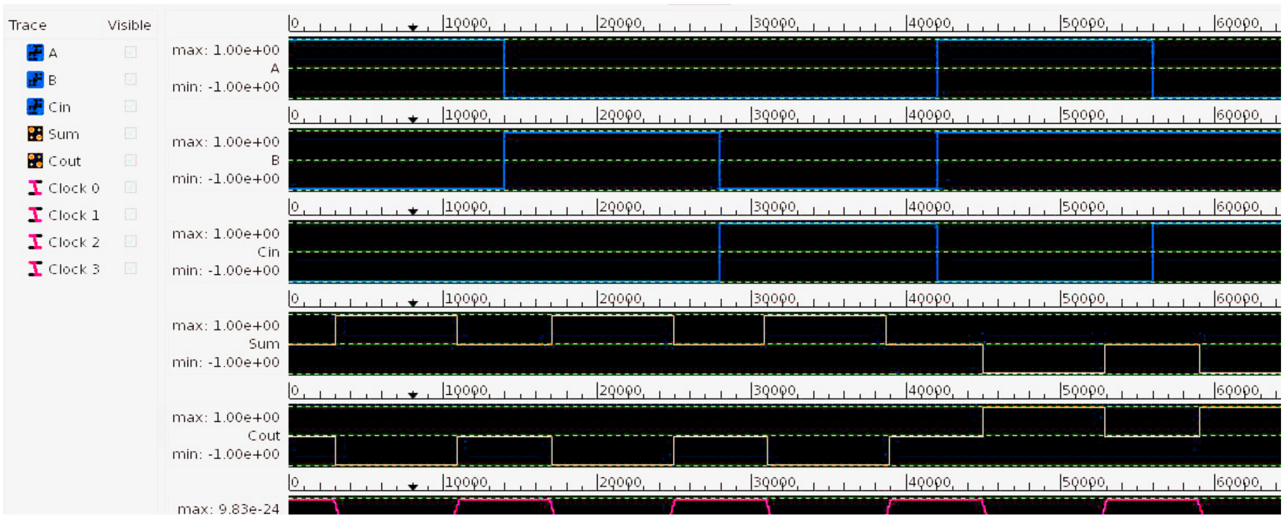


Figure 5.4 Simulation Results of 1-bit adder design using M-QCA Bistable Engine.

Notes: Other research efforts were made to create multiple-bit adders with different methods [12, 13, 14], readers may refer to them and create larger circuit designs.

6. Conclusion and Future Work

6.1 Conclusion

As the future of CMOS development is predicted to end soon, nanomagnetics can provide a new computing paradigm for information processing using the principles of magnetic quantum cellular automata (M-QCA). Energy dissipation can be reduced by many orders of magnitude compared to present-day CMOS technology [15].

M-QCA bistable simulation engine is able to simulate a large number of cells at rapid speed, and therefore provides a good, in process, check of a design. The bistable M-QCA simulation is sufficient to verify the logical functionality of a design, however the engine has to be extended to yield accurate dynamic simulation results.

6.2 Future Work on Coherence Engine

M-QCA bistable simulation engine considers the two final states of the system and solves the time independent case. For the coherent case, we need to model the time evolution of the magnetic elements making up the M-QCA cell based on the Landau-Lifshitz equation:

$$\frac{d\mathbf{M}}{dt} = -\gamma \mathbf{M} \times \mathbf{H}_{\text{eff}} + \lambda \mathbf{M} \times (\mathbf{M} \times \mathbf{H}_{\text{eff}}) \tag{6.1}$$

The first term in equation 6.1 describes the precessional motion of the magnetization under the influence of an external field, the second term describes the damping of the precessional motion toward the axis of the external field. HPLP group at UVa has developed a working matlab module of the dynamic motion of the magnetization transition which render Figure 6.1 below.

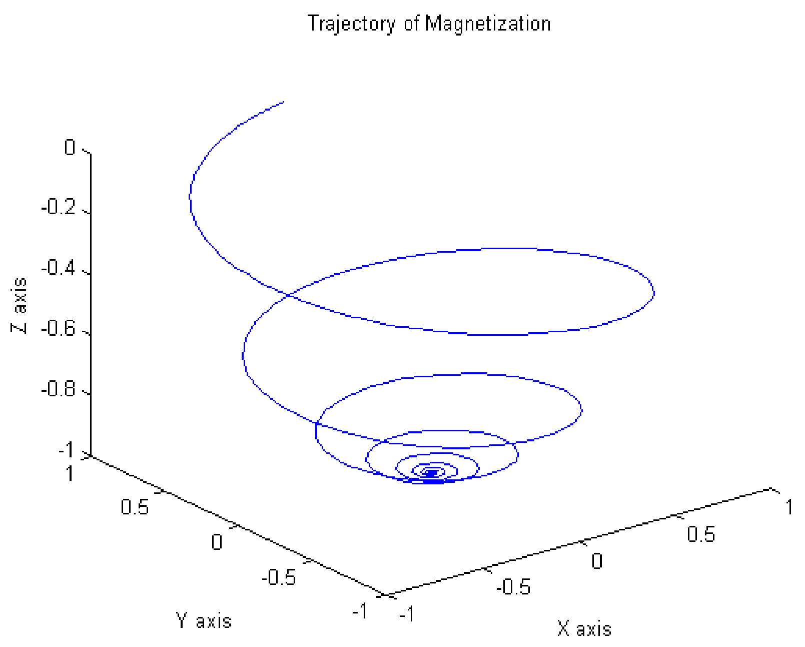


Figure 6.1 Dynamic trajectory of the magnetization of a single domain magnet in the presence of stress. Courtesy to M. Kabias HPLP group at UVa.

To model the dynamic case of M-QCA circuits as the next stage of M-QCA project, he/she needs to focus on replacing the bistable algorithm with the coherent case. This thesis and its accompanying documents can serve as a complete set of references for designing the programming and executing specific steps to implement the new simulator and integrate with the existing.

References

[1] R. P. Cowburn, D. K. Koltsov, A. O. Adeyeye, and M. E. Welland, “Lateral Interface Anisotropy in Nanomagnets”, Journal of Applied Physics, vol. 87, pp. 7067-7069, May 1 2000.

[2] M. E. Welland, R. P. Cowburn. “Room Temperature Magnetic Quantum Cellular Automata”, Science, 287: 1466-1468, 2000.

[3] G. H. Bernstein, A. Imre, V. Metlushko, A. Orlov, L. Zhou, L. Ji, G. Csaba, and W. Porod, “Magnetic QCA systems”, Microelectronics Journal, vol. 36, pp. 619-624, July 2005.

[4] M. Stan, M. Kabir, J. Lu and S. Wolf, “Self-Assembled Multiferroic Magnetic QCA Structures For Low Power Systems and Memories”, Proc. of GLSVLSI 2011, 25 - 30, May 2011

[5] F. Zavaliche, T. Zhao, F. Straub, M. P. Cruz, P. L. Yang, D. Hao, and R. Ramesh, “Electrically Assisted Magnetic Recording in Multiferroic Nanostructures”, Nano Letters, vol. 7, pp. 1586 - 1590, June 2007

[6] K.. Walus. “Design and Simulation of Quantum-Dot Cellular Automata Devices and Circuits”. PhD thesis, University of Calgary, June 2004.

[7] B. D. Cullity; C. D. Graham. “Introduction to Magnetic Materials, 2nd ed.”. New York: Wiley-IEte=2008. Pp. 116. ISBN 0-471-47741-9.

[8] Domb, C., 1974, "Ising Model", in Domb, C., and Green, M.S., 1974, Phase Transitions and Critical Phenomena, Vol. 3, 357-484.

[9] J. Huang, et, al. 2008, “Tile-Based QCA Design”, in Deisgn and Test of Digital Circuits by Quantum-DOT Cellular Automata, pp. 171-211

[10] R. Zhang, K. Walnut, Wei Wang and G. Jullien, “A method of majority logic reduction for quantum cellular automata”, IEEE Trans. Nanotechnology, vol. 3, no. 4, pp. 443-450, Dec. 2004.

[11] ATIPS Labs QCADesigner Tutorial. http://www.atips.ca/projects/qcadesigner/tutorial.html.

[12] Sansiri Haruehanroengra1, a and Wei Wang Solid State Phenomena Vols. 121-123 (2007) pp 553-556 online at http://www.scientific.net © (2007) Trans Tech Publications, Switzerland Online available since 2007/Mar/15

[13] Maryam Taghizadeh, Mehdi Askari, Khossro Fardad, BCD Computing Structures in Quantum- Dot Cellular Automata Proceedings of the International Conference on Computer and Communication Engineering 2008 May 13-15, 2008 Kuala Lumpur, Malaysia

[14] Heumpil Cho, Student Member, IEEE, and Earl E. Swartzlander, Jr., Fellow, IEEE Adder Designs and Analyses for Quantum-Dot Cellular Automata, IEEE TRANSACTIONS ON NANOTECHNOLOGY, VOL. 6, NO. 3, May 2007

[15] S. Mukhopadhyay, ‘‘Switching energy in CMOS logic: How far are we from physical limit?’’ 2006. [Online]. Available: http://nanohub.org/resources/1250.

[16] Philosophy of the gnu project, 2006. http://www.gnu.org/philosophy/philosophy.html.

[17] The gnu operating system, 2005. http://www.gnu.org

[18] Gnu general public license, 1991. http://www.gnu.org/licenses/gpl.txt

[19] Gnome: The free software desktop project, 2006. http://gnome.org/.

[20] Gimp - a brief (and ancient) history, 2003. http://www.gimp.org/about/ancient history.html.

[21] Gtk+ - the gimp toolkit. http://www.gtk.org

[22] Gnome/gtk+ software repository, 2006. http://www.gnomefiles.org/.

[23] QCADesigner API Reference. http://www.mina.ubc.ca/qcadesigner_api_reference

[24] G. Schulhof, “QCADesigner: From Utility To Application”, Master thesis, University of Calgary, October 2006

Appendix A: M-QCA User Guide

The complete package of M-QCA project files is available for download at http://www.hplp.com. In Root directory, *readme.txt* contains instructions for compilation. This new version of QCADesigner has the added-in M-QCA Bistable simulation engine outside of the original three QCA simulation engines. Detailed user manual of the QCA engines can be found at http://www.mina.ubc.ca/qcadesigner

This appendix is mainly discussing the M-QCA engine and its pertaining usages.

M-QCA Simulation Engine

The QCA computing paradigm is extremely efficient, but requires low temperature operation (~1 K) due to the current state of semiconductor technology. As a result, many research groups have invested numerous hours and effort exploring the field of implementing Boolean logic gates with magnetic dots interacting via dipole-dipole interactions.

Currently, there are many forms of M-QCA logic models; this M-QCA simulation model is based on the idea that we arrange a large and regular array of nanopillars in a square pattern, as illustrated in Figure A.1, which we call a cell bit.

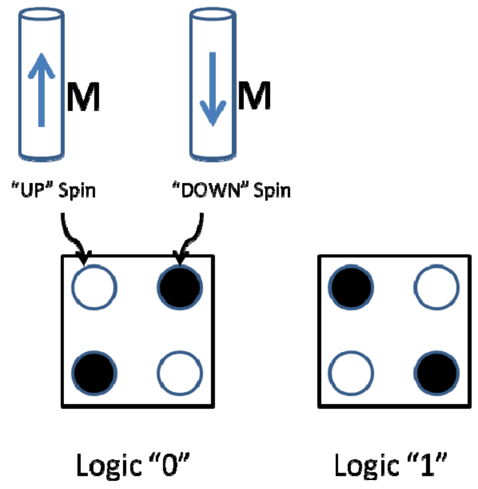


Figure A.1 Four Pillar Bit Pattern

Each cell bit consists of four pillars in a square with two corners polarized up and two down.

This M-QCA model is one of the closest to the QCA model in term of design and implementation. It also has the desired form for RAMA (Reconfigurable Array of Magnetic Automata) circuit designs. RAMA shares the same foundation with other M-QCA models, but differentiates itself by placing the magnetic bits in arrays in which the dots can be electrically addressed individually with electric fields instead of magnetic fields [1].

Kink Energy

Similar to other QCA model in QCADesigner, the kink energy $E_k^{i,j}$ represents the energy cost of cells i and j having opposite polarization; In another word, the magnitude of E_k determines how much of influence cell i's magnetization has on cell j.

For a general case of dipole-dipole interaction, please see Figure A.2 below.

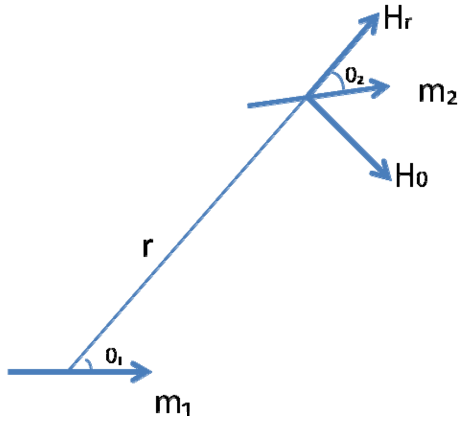


Figure A.2 Interacting dipoles

This figure illustrates two magnets of moment m_1 and m_2 at a distance r apart and making angles θ_1 and θ_2 with the line joining them. It can be shown [2] that the potential energy of m_2 in the field of m_1 is

$$E_p = \frac{-m_1 \cdot m_2}{r^3} \cdot (2 \cdot cos\theta_1 \cdot cos\theta_2 - sin\theta_1 \cdot sin\theta_2) \tag{A.1}$$

In another form

$$E_p = \frac{m_1 \cdot m_2}{r^3} \cdot (\cos(\theta_1 - \theta_2) - 3 \cdot cos\theta_1 \cdot cos\theta_2) \tag{A.2}$$

Similarly, it can be shown that the potential energy of m_1 in the field of m_2 is given by the same expression, and E_p is the mutual potential energy of the two dipoles, which is also called dipole-dipole energy or the interaction energy between the two dipoles, it is fundamentally magnetostatic energy.

In M-QCA bistable simulation engine, we exam the final states of the system, which means the pillars settle in either parallel or anti-parallel positions. Then the above equation can be simplified to

$$E_p = \frac{m1 * m2}{r^3}, \tag{A.3}$$

when the magnetization of the two pillars are in parallel, where $\theta_1 = \theta_2 = 90^o$, and,

$$E_p = -\frac{m1 * m2}{r^3}, \tag{A.4}$$

when magnetization settles in anti-parallel position, where $\theta_1 = -\theta_2 = 90^o$.

It is apparent that the system would prefer the anti-parallel position, as that is when the overall energy state is at the minimum.

To calculate kink energy between cell *i* and *j*, for each pillar in cell *i* we need to calculate the dipole-dipole interaction between the pillar and each pillar in cell *j*, then sum over all *i* and *j*. Specifically, we need to first calculate the energy when the cells have opposite magnetizations, then when the cells have the same magnetizations, and subtract the two.

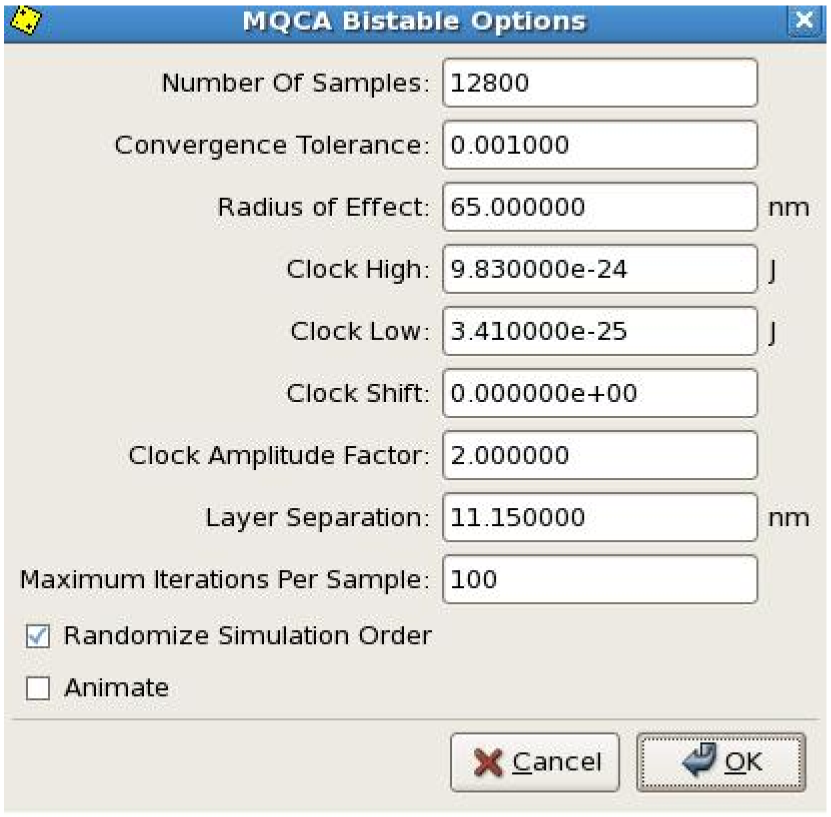
Magnetic Bistable Engine

For the two-state system, it has the same Hamiltonian structure as QCA except for there will not be tunneling energy in between neighboring pillars. The final magnetization relation can be reduced to the following:

$$M_i = \frac{E_k^{i,j} * \left(\sum_j M_j\right)}{\sqrt{1 + E_k^{i,j} * \left(\sum_j M_j\right)}} \tag{A.5}$$

Where M_i is the magnetization of the cell, and M_j is the magnetization state of the neighboring cells within a preset effective radius. Once the circuit has converged, the output is recorded and new input values are set. The bistable M-QCA simulation is sufficient to verify the logical functionality of a design; however a separate coherence engine is need for continuous dynamic simulations. As a result of its simplicity, M-QCA bistable simulation engine is able to simulate a large number of cells at rapid speed, and therefore provides a good, in process, check of a design.

Engine Options


MQCA Bistable Options

Number Of Samples: 12800

Convergence Tolerance: 0.001000

Radius of Effect: 65.000000 nm

Clock High: 9.830000e-24 J

Clock Low: 3.410000e-25 J

Clock Shift: 0.000000e+00

Clock Amplitude Factor: 2.000000

Layer Separation: 11.150000 nm

Maximum Iterations Per Sample: 100

✓ Randomize Simulation Order

☐ Animate

Figure A.3 Dialog box of 'MQCA Bistable Options'

Number of Samples

The total simulation is divided by the number of samples. For each sample, the simulation engine looks at each cell and calculates its magnetization based on the magnetization of its effective neighbors (determined by the radius of effect). The larger this number the longer the simulation will take to complete. Choosing an appropriate number is important for getting correct output.

For M-QCA cases, the energy magnitude is higher than QCA, thus to ensure an adiabatic switching, we need to slow down the clock switching time. Unfortunately, in the current version of all the engines in QCADesigner, it is not a parameter one can set directly. Indirectly, the clock frequency is associated with number of samples due to the build-in calculation algorithm.

TIP: Through experiment, it is discovered that the number of Samples should be set around 700000 to ensure a correct results for the coplanar crossing, which corresponding to a clock switching time of 0.33 ps. Thus for M-QCA circuit simulation, it takes longer simulation time to complete.

Convergence Tolerance

During each sample, each cell is converged by the simulation engine. The sample will complete when the magnetization of each cell has changed by less than this number; i.e. loop while any design cell has (old_magnetization - new_magnetization) > convergence_tolerance.

Radius of Effect

Because the interaction effect of one cell into another decays inversely with the distance between cells, we do not need to consider each cell as effecting every other cell. This number determines how far each cell will look to find its neighbors. Make sure that at least the next-to-nearest neighbors are included in this radius. If you only allow for the nearest neighbor then I would not expect designs with coplanar crossovers to work. The following figure should help clear this up.

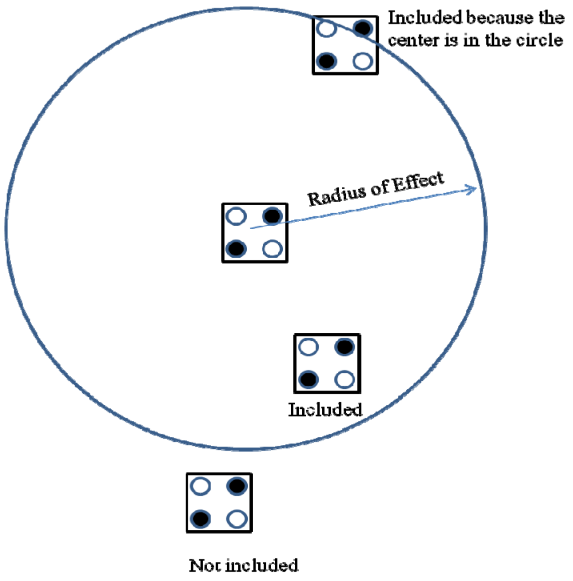


Figure A.4 Radius of effect illustration diagram, courtesy to MiNa group at UBC.

Note that with multilayer capability the radius of effect is extended into the third dimension. Therefore in order to include cells in adjacent layers, make sure that the separation layer is less than the radius of effect.

Clock Signal

The clock signal in **QCA*Designer*** is calculated as a hard-saturating cosine as shown below in Figure A.5.

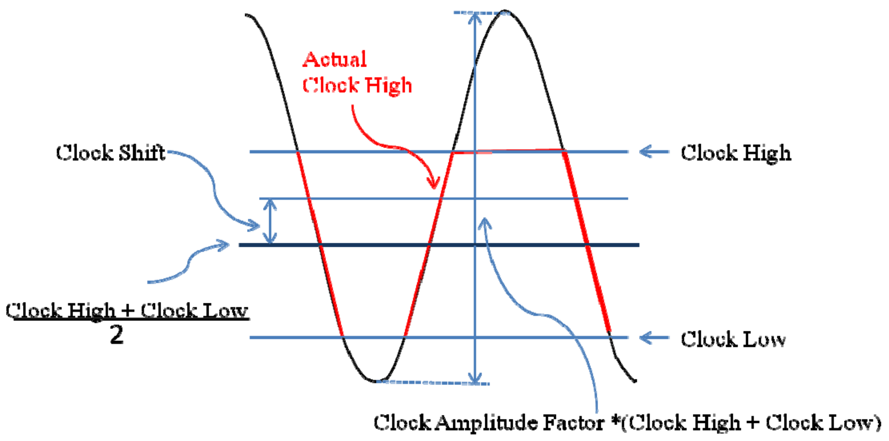

Actual Clock High
Clock Shift
Clock High
Clock High + Clock Low / 2
Clock Low
Clock Amplitude Factor *(Clock High + Clock Low)

Figure A.5 Clocking Options and illustration diagram

Clock High/Clock Low

Clock low and high values are the saturation energies for the clock signal. When the clock is high the cell is unlatched. When it is low the cell is latched. The defaults seem to work in most cases; however, you may wish to play with them.

Clock Shift

The clock shift allows you to add a positive or negative offset to the clock as shown in the above figure.

Clock Amplitude Factor

Clock amplitude factor is multiplied by (Clock High - Clock Low) and reflects the amplitude of the underlying cosine as shown in the above figure.

Layer Separation

When simulating multilayer QCA circuits, this determines the physical separation between the different cell layers in [nm]. Cell in the adjacent layer assumes the opposite logic function, thus one need odd number of via layer and one crossover layer to get the correct information transferred to the other end of bridge. Generally, I found use only 1 via layer and 1 crossover layer is sufficient to get the information crossed with the radius of effect and layer separation variables set to 45 nm and 20 nm respectively.

Maximum Iterations Per Sample

If the design does not converge in this number of iterations, then the simulation will move on to the next sample point. For complex designs you may wish to increase this.

Randomize Simulation Order

When active, the order in which cells are simulated is randomized in each iteration. It is recommended to leave this on.

Note: This version of QCADesigner is covered under the GNU Public License, distribution and modification of the codes should be conducted under these terms.

References

[1] M. Stan, M. Kabir, J. Lu and S. Wolf, “Self-Assembled Multiferroic Magnetic QCA Structures For Low Power Systems and Memories”, Proc. of GLSVLSI 2011, 25 - 30, May 2011

[2] B. D. Cullity; C. D. Graham.”. New York: Wiley-IEte=2008. pp. 116. ISBN 0-471-47741-9

Appendix B: Historic Context of Tools and Libraries

Figure below illustrates the containment structure of the acronyms of the various projects related to QCADesigner:

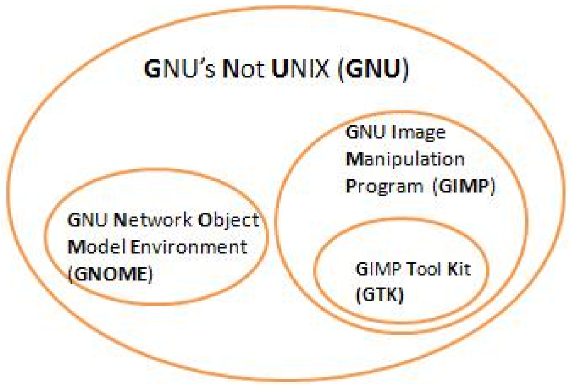

GNU's Not UNIX (GNU)
GNU Image Manipulation Program (GIMP)
GNU Network Object Model Environment (GNOME)
GIMP Tool Kit (GTK)

Figure B.1 Acronyms and Relationships among the open source projects.

B.1.1 GNU

The GNU operating system is a complete free software system, upward-compatible with Unix. GNU stands for “GNU's Not Unix”. Richard Stallman made the initial announcement of the GNU Project in September 1983. A longer version called the GNU Manifesto was published in March 1985. The word “free” in “free software” pertains to freedom [16], not price. You may or may not pay a price to get GNU software. Either way, once you have the software you have four specific freedoms in using it. The freedom to run the program as you wish; the freedom to copy the program and give it away to your friends and co-workers; the freedom to change the program as you wish, by having full access to source code; the freedom to distribute an improved version and thus help build the community.

The GNU Project [17] was conceived in 1983 as a way of bringing back the cooperative spirit that prevailed in the computing community in earlier days—to make cooperation possible once again by removing the obstacles to cooperation imposed by the owners of proprietary software.

This freedom mentality is quantified by the license where under GNU software packages (and, in recent times, many other software packages not a part of the GNU project) are released: The GNU Public License (GPL) [18]. QCADesigner is released under the GPL, so is the new version with M-QCA bistable simulation engine.

Historically, GNU has been used on many proprietary UNIX-like operating systems in conjunction with their own, proprietary tools. However, at the beginning of the 1990s, the Linux operating system kernel was written by Linus Torvalds. It was also released under the GPL. Combining Linux with the almost-complete GNU system resulted in a complete operating system: the GNU/Linux system. Estimates are that tens of millions of people now use GNU/Linux systems, typically via GNU/Linux distributions.

B.1.2 GNOME

Linux's UNIX-like nature manifests itself in its flexibility. Unfortunately, for users who are not computer experts, the price for flexibility comes in the form of user-unfriendly. Therefore a graphical desktop (called GNOM) is developed to help beginners use the GNU system. The GNOME project [19] was started in 1997 by Miguel de Icaza with the purpose of creating a usable, consistent, stable, and efficient desktop environment for Linux. This would provide end users with a familiar, architecture-independent and consistent portal for performing their tasks, and it would provide developers and distributors with an opportunity to offer a consistent and familiar user interface for the solutions their distributions offered.

B.1.3 The GIMP

The GNU Image Manipulation Program [20] began as a class project for two University of California at Berkeley students, Spencer Kimball and Peter Mattis. It gained momentum through the addition of plug-ins and, eventually, scripting languages, but mostly through its adoption by a large user base.

To display its graphical user interface, the GIMP initially used the Motif toolkit. However, as it matured, its developers found Motif to be to unstable and decided to create their own toolkit: GTK.

B.1.4 GTK

The original purpose of The GIMP Toolkit [21] was to provide a custom user interface for the GIMP. However, when it became the user interface toolkit of choice for the GNOME project, many applications started using it as their user interface toolkit both from the outset and after switching away from another. This broad adoption in turn caused those who used GTK for their own purposes to contribute refinements and bug fixes back to the GTK developers.

Today, GTK+ (as GTK became known) relies also on Pango (a font rendering library), and ATK (the accessibility toolkit - for persons with disabilities) to provide a highly customizable and consistent user interface. A large number of projects employing GTK+ and/or GNOME libraries, including QCADesigner, are showcased online [22].

Appendix C: Inherited Data Structures and Objects

C.1 QCADCell

The QCADCell structure is updated for M-QCA model: a variable magnetism is inserted in QCADCellDot structure because its magnitude is the new measurement for assessing cell logic instead of polarization in M-QCA simulation engine.To see a complete list of data structures used in the development of other originial QCA simulation engines, refer to [23].

QCADCell is one descendant of QCADDesignObject representing a cell. All the characteristics of QCA cells listed below are applicable for M-QCA cells. Most of the contents are cited from [24] with minor modifications in order to help M-QCA readers to speed up the ramp-up time dissecting the codes.

One cell assumed to consist of 4 dots arranged either in the corners of a rectangle (normal cells), equidistant from the center of the cell, or at the midpoints of the sides of a rectangle (rotated cells).

QCA/M-QCA cells can have four different functions:

• A “normal” cell is one affected by the circuit’s clocking arrangement and by neighbouring cells. In other words, it behaves “normally”.

• An “output” cell is identical to a normal QCA cell, however, during simulation, the polarization values it takes on are recorded as part of the simulation data.

• An “input” cell behaves independently of its neighbours and of the circuit’s clocking arrangement. Its value is assigned at the beginning of a simulation iteration based on the current sample index either for an exhaustive verification or from a vector table.

• A “fixed” cell maintains a given polarization irrespective of clock values or the values of its neighbors. Such cells are necessary, because not all classical logic circuits can be mapped to QCA circuits consisting of majority gates whose inputs are exclusively non-constant.

C.2 VectorTable

The vector table data structure (VectorTable) consists of:

• the string pszFName indicating the path on disk containing the vector table

• A 2D array of vectors, implemented using EXP_ARRAY (see Section C.3)

• an array of VT INPUT structures, each of which contains an input QCADCell, as well as a boolean flag indicating whether the input is active (active flag). An inactive input is treated by the simulation engine like a normal cell, influenced by the polarizations of its neighbors and by the clocks.

C.3 EXP_ARRAY

An EXP_ARRAY is similar to a GArray offered by GLib in that it can grow to accommodate an arbitrary number of values on demand. There are, however, some key differences in the intended usage of the respective structures and, consequently, in their respective implementations.

C.3.1 Dimensionality

GArray was intended to be a single dimensional array. It does not, for example, provide a constructor allowing one to specify the desired dimensionality of the array. In contrast, EXP_ARRAY was designed to be multidimensional.

A multidimensional EXP_ARRAY is essentially an EXP_ARRAY structure consisting of EXP_ARRAYs, forming a tree such that its depth is equal to the number of dimensions. Then, indexing into the array is simply the process of indexing into the array located at each level to retrieve the array representing the next level of indices until, at the leaf nodes, the data is accessed.

Space for data items in an EXP_ARRAY is not preallocated. Thus, arbitrary indices cannot be accessed until they are first stored. Any additions to an EXP_ARRAY using the provided functions will cause the addresses placing these additions to be adjusted such that the added items are either inserted between existing items or appended to the end of the array.

C.3.2 Monotonically Increasing Allocation

EXP_ARRAY allocates memory in a monotonically increasing fashion. That is, memory previously allocated for the addition of new items is not de-allocated when those items are removed. New memory is allocated if the total space required for new additions exceeds the total available space, including recyclable space. In the case of multidimensional arrays, entries are removed from non-leaf arrays when they become empty, and any succeeding entries are moved up. The unlinked arrays are freed.

C.3.3 Implementation

<u>Data Structure</u>

EXP_ARRAY consists of 5 items:

• data is a pointer to the array data

• icUsed is a count of valid elements currently stored at data

• icAvail is a count of available slots before the memory allocated at data needs to be increased

• cbSize holds the size of an individual element

• icDimensions holds the dimensionality of this array

<u>Creation</u>

The constructor for an EXP_ARRAY takes two arguments: The element size and the dimensionality. It allocates the memory for the structure itself and assigns cbSize and icDimensions. It sets both counters (icUsed and icAvail) to 0, and sets data to NULL.

<u>Destruction</u>

EXP_ARRAY structures are destroyed recursively. If a given array's dimensionality is larger than 1, then its data is itself an array of EXP_ARRAY structures. These are freed with recursive calls. Thereafter, data is freed. The destructor's last action is to free the EXP ARRAY structure itself.

<u>Insertion</u>

The function inserting elements into an EXP_ARRAY takes at least three arguments (aside from the array itself): data is a pointer to a conventional array containing the data to be inserted. icElements is the number of elements stored at data.

iDimension is the dimension where they are to be inserted. A sequence of pairs of indices and booleans, equal in number to the dimensionality of the EXP_ARRAY follows.

Due to the mechanics of C functions with variable length argument lists, the insertion function visible to callers is a wrapper function for the one implementing the actual insertion. Before calling the internal insertion function, the wrapper clamps the iDimension passed by the caller between 1 and the number of dimensions in the EXP_ARRAY. This eliminates invalid values. The private insertion function retrieves an (index, boolean) pair from the variable argument list. The boolean bInsert accompanying the index idx indicates whether the function is to force the creation of an array entry at idx (bInsert is TRUE) or

whether it will attempt to reuse an existing index (bInsert is FALSE). As long as iDimension is greater than 1, the function will recurse, passing itself iDimension - 1. When iDimension reaches 1, the function is in one of two states:

• It is at a non-leaf node. In this case, it will loop over data, recursing with each one of its elements at-a-time over the child EXP_ARRAY structures, using the remaining indices.

• It is at a leaf node. In this case, data can be inserted directly, as the data member of the EXP_ARRAY points to an array of the same type. In this case, bInsert is ignored and assumed to be TRUE, to avoid overwriting existing data members.

Removal

Elements can be removed from an EXP_ARRAY in groups forming n-dimensional parallelepipeds, where n is the dimensionality of the array. This is accomplished by supplying the removal function with n tuples of (start index, count) integers, specifying the starting index and the element count for each dimension. Of course, for each dimension, both values are clamped to the range of available indices and element counts, respectively. For the element count this means that, if idx + icElements exceeds the value of the array's icUsed member, then icElements is adjusted to remove all elements from idx inclusive, to the end of the array.

For each dimension except the last one (the leaf nodes), the removal function accomplishes its task in two passes. In the first pass, it loops from its tuple's start index idx as many times as the count specifies (icElement times) and recurses with each child EXP_ARRAY found and the rest of the tuples. In the second pass, looping from idx to the end of the array, it searches for those child arrays that have become empty as a result of the recursive removal. When it finds such an array, it frees it, and it moves arrays with larger indices up to fill the gap. Thus, the array elements stay contiguous. The highest index value reached by the second pass is 1 smaller than the number of elements in the array. Thus, the icUsed member of the array is set to a value 1 higher than the highest value of the index Nix used in the second pass.

For the leaf nodes, the function simply results in a memmove, moving whatever data items remaining at idx + icElements up to idx. icElements is subtracted from the icUsed member of the array.

<u>Convenience</u>

QCADesigner's EXP_ARRAY library includes several convenience items, in the form of C preprocessor macros, as well as convenience functions. The preprocessor macros are:

• exp array index 1d indexes into a 1D_EXP_ARRAY.

• exp array index 2d indexes into a 2D_EXP_ARRAY. The 2 indices form a (row, column) tuple.

• exp array 1d insert vals is a shortcut for inserting values into a 1D array. Since the iDimension parameter of the insertion function is always equal to 1 and the bInsert parameter in the first and only tuple always evaluates to TRUE, this macro renders insertion into a 1D_EXP_ARRAY more readable in appearance.

The convenience functions are:

• exp_array_1d insert val sorted takes a data element data and inserts it into the array based on a partial order defined by the function pointer fn and the parameter bAllowDupes. This function's behaviour on unsorted arrays is undefined. However, if, from instantiation, elements are inserted into the

array using only exp array 1d insert val sorted, a sorted array will result. bAllowDupes determines whether values deemed equal by the partial order are stored alongside one another, or whether an attempt to insert a duplicate is rejected. The function returns the array index of the new element or, if the

element was rejected, −1.

• exp array 1d find is a binary search that can be performed on a sorted array. It accepts an element being sought, a partial order, and a parameter bClosest indicating whether to return the closes match found in case an exact match is not found. The actual implementation of the binary search is done by a private function that always returns an index between 0 and the number of elements in the array (the icUsed member). The public function interprets this value according to bClosest and returns a valid index to the caller, or −1 in case a match is not found.

• exp_array_copy performs a shallow copy of an array.

• exp_array_dump prints out the contents of an array using hexadecimal notation. It uses reverse video to indicate which items are within icUsed. The items between icUsed and icAvail are printed normally.

C.4 BUS_LAYOUT

The data structure encoding the bus layout for a design (BUS LAYOUT) consists of three EXP_ARRAYs (as described in Sec. C.3):

• inputs

• outputs

• buses: inputs and outputs are arrays of BUS LAYOUT CELL structures. BUS LAYOUT CELL

structures consists of:

• A QCADCell

• A boolean variable bIsInBus indicating whether the cell is currently in use by one of the buses

buses is an array of BUS structures.

Each BUS structure consists of:

• A string pszName holding the name of the bus

• A QCADCellFunction flag named bus function set to either QCAD CELL INPUT or QCAD CELL OUTPUT

• An array of integers named cell indices, indices into one of inputs or outputs, depending on bus function inputs and outputs reflect the current design's contingent of inputs and outputs.

They are kept up-to-date by handling cell layers' "added" and "removed" signals. buses reflects the user's choice in grouping those inputs and outputs into buses. The user determines the order of the buses in buses, as well as the order of the bits inside a bus (inside cell indices) by means of the bus layout dialog.

C.5 simulation_data

simulation_data structure stores simulation results later used for displaying. It consists of an integer number_samples denoting the number of samples in each of the structure's traces. The number_of_traces is the number of traces in the trace array. In addition, there are four traces contained in the clock_data array, each representing one of the clocks.

The array of traces as well as the array of clocks consists of TRACEDATA structures, each representing the waveform of a previously simulated output cell. Each TRACEDATA structure contains a string, data labels. This is the name of the input or output cell the waveform refers to. The trace function member is set to either QCAD CELL INPUT if the waveform refers to an input, QCAD CELL OUTPUT if the waveform refers to an output, or QCAD CELL FIXED if the waveform refers to a clock. The boolean drawtrace member controls whether the trace is to be displayed. data is an array of number samples double-precision floating point values, holding the value of the waveform at each recorded sample.