Project Delta
An Interactive FPGA Circuit Simulator

Specification

Developers:
Robert Duncan – `rad55@cam.ac.uk`
Justus Matthiesen – `jm614@cam.ac.uk`
David Weston – `djw83@cam.ac.uk`
Christopher Wilson – `cw397@cam.ac.uk`
Rubin Xu – `rx201@cam.ac.uk`

Client:
Steven Gilham
`steven.gilham@citrix.com`

January 27, 2009

Contents

1 Project description

It would be useful if a first year computer scientist could rapidly prototype a circuit on an Altera DE2 FPGA board without using a hardware description language. The user interface might be presented as a graphical circuit entry system allowing the user to select components from a library (including, 2- and 3-input NOR and NAND gates, RS-latch and D flip-flop) and place them on the screen. Additional components like RAMs and ROMs would be desirable. Switches and LEDs that are present on the DE2 board should be present in the graphical environment to allow circuits to be interfaced to them. Saving and loading of circuits is desirable. Circuits created should be simulated on the DE2 board. The transfer of the circuit to the FPGA simulation might happen in real-time (i.e. any change in the circuit is reflected "instantly" in the simulation). The simulation might be performed in software by a soft processor on the FPGA. For a high performance implementation, the graphical circuit might also be written out as Verilog and then synthesised for direct implementation on the FPGA.

2 Proposal

A Java based solution is proposed with a GUI that allows the user, in this case a first year computer scientist, to design a simple circuit using a drag-and-drop interface. The application will allow the user to send their design to the Altera DE2 board for simulation using a Java-based processor. What follows is a description of the required features and our plan to implement them within the time restriction given.

3 Major planned features

(*a*) Tri-state wires: with `0`,`1`, and `X` states.

(*b*) A component library containing:

- (*i*) NOR gate (2/3 input).
- (*ii*) NAND gate (2/3 input).
- (*iii*) AND gate (2/3 input).
- (*iv*) OR gate (2/3 input).
- (*v*) XOR gate (2/3 input).
- (*vi*) XNOR gate (2/3 input).
- (*vii*) NOT gate
- (*viii*) Fixed Input (`0`/`1`)
- (*ix*) RS latch.
- (*x*) D flip-flop.
- (*xi*) limited size RAM.

(*xii*) limited size ROM.

(*c*) The ability to "connect" circuit to built-in LEDs, toggle switches, and push buttons on DE2 boards.

(*d*) The ability to group and ungroup components into a single composite component.

(*e*) Reasonably accurate simulation of circuit on DE2 board, not withstanding variable gate and wire delay (i.e. only have fixed gate delay equal across all components).

(*f*) Single clock with variable frequency.

(*g*) Components can have multiple wires attached to each output connector, however they may only have one wire for each input connector.

(*h*) An intuitive GUI with:

  (*i*) Expandable component library.

  (*ii*) Switch/button/LED library.

  (*iii*) Drag-and-drop component placing and wiring.

  (*iv*) Flexible wiring that can attach to input/output connectors on components.

  (*v*) Undo/redo.

  (*vi*) Copy/paste.

  (*vii*) Zooming.

  (*viii*) Document loading/saving.

(*i*) Capability to export to verilog to be implemented directly onto the FPGA.

(*j*) Fast transfer of designed circuit to DE2 board to be simulated.

4 Acceptance criteria

Even though we will endeavour to complete all the major features listed in section 3, we may find it necessary to make variations during development of the project. The final project must satisfy the following criteria:

(*a*) Core GUI.

(*b*) Loading/saving of circuits.

(*c*) At least switch/button/LED connection along with NAND and NOR gates (2/3 input), RS latches and D flip-flops.

(*d*) Simulation of circuit on DE2 board.

(*e*) Clock frequency control.

5 Circuit description

A limited number of components are going to represented, trying to fairly represent the knowledge and interests of a first year computer scientist.

We will be considering the output of all gates to initially be `X`, and all disconnected wires will also have a value of `X`. This tri-state approach is necessary to model disconnection of wires, and also to represent an unknown signal on a wire.

Here follows a description of each component represented in the circuit simulator:

5.1 Simple gates

5.1.1 NOT gate

A NOT gate is essentially an inverter, negating the input as per this table:

x	NOT(x)
0	1
1	0
X	X

where x is the input to the NOT gate.

5.1.2 AND and NAND gates

A two input AND can be desribed with the following table in our tri-state logic:

AND	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

A three input AND could be modelled as $y = ((x_1 \texttt{ AND } x_2) \texttt{ AND } x_3)$ where x_n are inputs to the AND gate, and y is the output.

A NAND gate has exactly this behaviour except the output, y, is inverted, as if it were by a NOT gate.

5.1.3 OR and NOR gates

A two input OR can be desribed with the following table in our tri-state logic:

OR	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

A three input OR could be modelled as $y = ((x_1 \texttt{ OR } x_2) \texttt{ OR } x_3)$ where x_n are inputs to the OR gate, and y is the output.

A NOR gate has exactly this behaviour except the output, y, is inverted, as it were by a NOT gate.

5.1.4 XOR and XNOR gates

A two input OR can be desribed with the following table in our tri-state logic:

XOR	0	1	X
0	0	1	X
1	1	0	1
X	X	1	X

A three input XOR could be modelled as $y = ((x_1 \texttt{ XOR } x_2) \texttt{ XOR } x_3)$ where x_n are inputs to the OR gate, and y is the output.

An XNOR gate has exactly this behaviour except the output, y, is inverted, as if it were by a NOT gate.

5.2 Further components

5.2.1 Clock

Each circuit will have a single clock associated with it, that is automatically tied to each clock input on any clocked components. The Clock will have a customisable frequency that is some multiple of the simulation time quantum.

5.2.2 D flip-flop

A D flip flop can be represented using a state transition table, where Q' represents the output Q after a clock tick:

D	Q	Q'
0	Q	0
1	Q	1
X	Q	X

We will be using single output D flip-flops, i.e. they will not contain an extra negated output. This doesn't affect the potential set of circuits though, since a user can attach multiples wires to a single output.

5.2.3 RS Latch

This a grouped component described by the following table:

R	S	Q	$\overline{Q}$
0	0	Q	$\overline{\texttt{Q}}$
0	1	1	0
1	0	0	1
1	1	U	U
0	X	X	X
X	0	X	X
1	X	X	X
X	1	X	X
X	X	X	X

Where `U` stands for unstable. This conbination of inputs is to be avoided if at all possible.

5.2.4 ROM and RAM

Whilst we don't provide the ability to use buses natively, they can be constructed from groups of wires. With both ROM and RAM the user will be able to preset the contents in the GUI before loading it to the DE2 board.

However, ROMs will not have the ability to set the contents of the cell when addressing it, only to read out the contents.

(a) 16 cells, addressable with 4 address inputs.

(b) Each cell stores a single boolean value.

(c) RAM has a write enable input along with a data input.

(d) A single output wire with the data contained at the addressed wire.

(e) an implicit clock connection, so that reads and write only occur on clock pulses.

5.3 Inputs and outputs

5.3.1 LEDs

These can be used within the circuit and they correspond to a labelled LED on the physical hardware.

5.3.2 Switches and buttons

These can be used within the circuit and they correspond to the labelled switch or button on the physical hardware.

6 Data structure and algorithm

The target audience for the proposed digital logic simulator is computer science students in their first year. The first year of a computer science syllabus typically contains a course in Digital Logic that treats the subject on a functional or gate level and is certainly not too concerned about physical details of logic circuits like wire delays, voltage drops due to wire resistance, etc... Thus, it has been decided to keep the simulation as close as possible to the functional level, where gates represent boolean functions from input signals (or wires in this model carry boolean values) to output signals. However, it has also been decided that the simulator should be capable of simulating any valid digital circuit.

To achieve a good compromise between both goals, the simulator will impose certain restrictions on circuits and the simulation will also model some digital circuit characteristics that go beyond pure functional simulation.

Restrictions on the data structures:

(*a*) If the user is allowed to create wires which connect two wires, it is possible to create circuits which only a simulator that models the physical properties of a circuit very accurately can simulate. The simplest example is a circuit where two wires carrying different logic signals are joined, see figure 1. To avoid such situations in the chosen

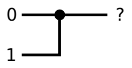

Figure 1: Indeterminable output.

model, wires are only allowed between an input of a logic component and an output of a logic component which are not necessarily different components. Furthermore, inputs to logic components are only allowed to be connected to one wire. This restriction does not in any way limit the number of different circuit the user can build but it does prevent the user from building circuit that cannot be simulated on a gate or functional level.

(*b*) Only one global clock is allowed which should be appropriate as only few circuits need more than one clock and even complex circuits, for example a microprocessor, can be implemented with only one clock.

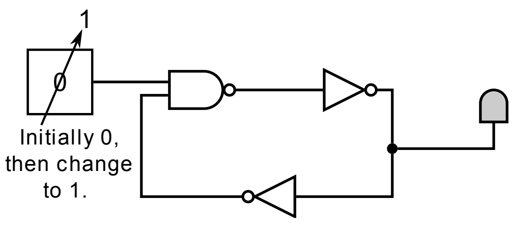

Figure 2: Inverter ring.

(*c*) A purely functional simulation would not emulate gate delays. The proposed simulator does not, however, disallow cycles in the circuit. Since cycles are allowed, circuits which are not stable, i.e. oscillate, can be constructed. The classic example is a ring of three inverters,see figure 2, the depicted example uses a NAND gate instead of an inverter so that the initial configuration produces deterministic results. Without gate delays this circuit cannot be simulated, at all, and therefore the simulation model needs gate delays. To stay close to a functional simulation the algorithm will use a unit-time gate delay for all gates.

(*d*) There are certain circuits that exhibit a behaviour that cannot be described in terms on combinatorial logic, that is they produce metastable outputs. Since it is algorithmically hard to detect whether a circuit belongs to this class of circuits and since this class includes components which are essential for sequential logic like RS latches, see figure 3, this class of circuits must not be rejected by the simulator. To simulate such circuits deterministically, a third logic state `X` in addition to `0` and `1`, representing logic states which are unknown or cannot be inferred by boolean logic.

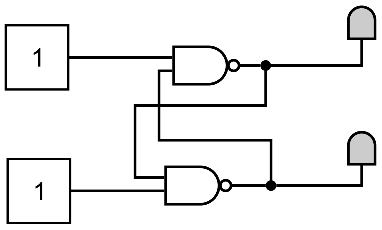

Figure 3: Latch.

6.1 Data Structure

The natural choice for a circuit representation is a graph-like data structure where vertices represent logic components and edges represent wires between component inputs and component outputs. Components are sub-circuits consisting of gates connected by wires, potentially represented by a graph-like data structure. Allowing circular circuits and the inclusion of gate delays make it, however, impossible to simulate a circuit component-wise. Thus, to allow editing the circuit in the GUI on a component level on the one hand and simulating the circuit on a gate level on the other hand, a data structure which can represent both has been designed: The data structure maintaining the circuit will hold two graphs simultaneously: One graph, in which vertices represent logic components, and one graph, in which vertices represent gates. When the user adds a logic component to the circuit, the component is first added to the former graph, the internal representation of the component consisting of gates and wires is then read and added to the latter graph (for more detail, see the UML diagrams in section 10.3).

6.2 Algorithm

For performance reasons, the simulation algorithm will be event-driven, i.e. the actual simulation algorithm is only run when an input signal changes. Inputs are defined in the application as buttons and switches on the board, constant input `0` and `1`, and the clock provided by the simulator. Switches and buttons have to be polled by the simulator at unit-time intervals to simulate event driven hardware inputs.

When an input signal changes, the algorithm sets the value, that each of the wires that is connected to the input carries, to the new logic value and saves the list of gates, or rather inputs to gates, that are connected by one of those wires to this input. And after a fixed unit-time gate delay, the algorithm re-evaluates the output of each of the gates in the list and propagates all output values, that have changed, in a similar fashion. To schedule such input signal changes and re-evaluations of gate outputs, the algorithm keeps a priority queue.

7 Using the Altera DE2 board

7.1 Overview

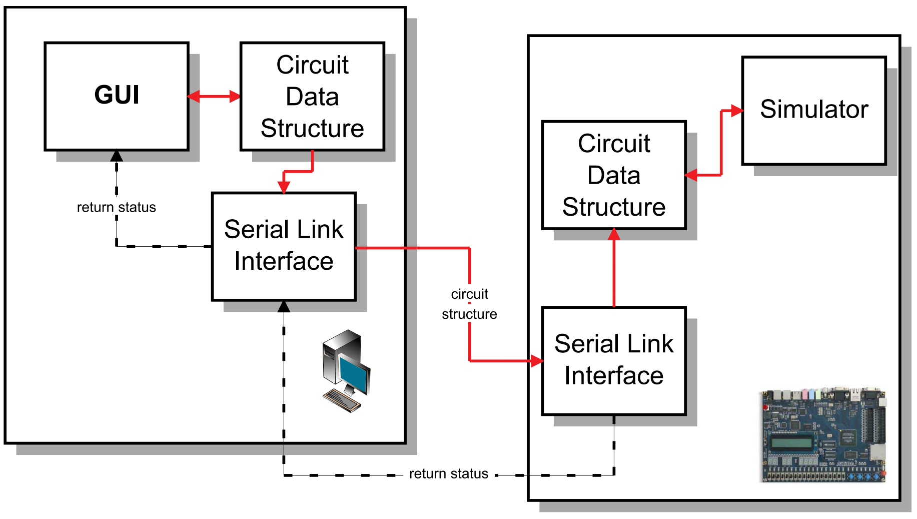

Figure 4: Overview of application structure.

7.2 Java processor

The simulation program will be written in Java and executed on an open-source Java bytecode processor, *JOP*[1], running on the Altera DE2 board. The *JOP* is a RISC processor with its own stack-based microcode instruction set. Java bytecode is translated into the corresponding microcode(s) during the decode stage of the processor cycle. A real-time garbage collector is implemented in Java, compiled as part of the target Java application and is invoked by the processor when necessary. All IO devices are integrated as memory-mapped devices into the uniform address space, distinguishing themselves from RAM spaces using the highest two bits of the address line. *JOP* provides a pre-existing toolchain to compile and run Java programs onto the DE2 board, thus enabling more time to spent on the GUI and the simulation program that will run on the board.

[1]JOP: A Tiny Java Processor Core for FPGA `http://www.jopdesign.com/`

Access to the DE2 board's LEDs and switches can be made through memory-mapped IO. Simple Java wrapper classes can be used to present an abstracted interface to the simulation program.

Access to the SDRAM provided by the board (8MB) can be provided with a adapted open-source SDRAM controller[2] for the Altera DE2 board. This access will be multiplexed with the access to the faster SRAM (512KB) to provided a unified address space that can be access from a Java interface.

7.3 Communication

A serial connection is already provided for within the *JOP* toolchain, a datagram layer can be adapted from the JOP libraries to provide a packetised interface to and from the board that is suited to the needs of the application. On top of this datagram layer, a serialized object channel will be used to communicate the circuit design from the GUI to the simulation program running on the board.

There is the possibility of emulating the serial connection using the USB blaster connection to avoid having two cables between the board and the computer used for the design work. However, at the moment only one-way connection (to the computer from the board) on the USB cable has been achieved.

This section contributed by Rubin Xu.

8 Exporting to Verilog

Exporting the circuit description to Verilog means that our GUI application to design circuits can be used independently of the DE2 board. This enables a huge variety of hardware devices to be used with our application other than the DE2 board. It also allows for faster circuit simulation than using the *JOP* on the board, allowing for more realistic simulation.

Verilog modules will be used to describe the D flip-flop, RS latch, RAM, and the ROM components. Using structural Verilog to describe the structure of the circuit, the application will be able to quickly produce Verilog descriptions from the internal data structure described in section 6.

We will not implement a software component to send the Verilog to the board within the application, instead producing a Verilog `.v` source file to retain greater flexibility in the destination of the Verilog description.

9 GUI

The user interface will consist of the following elements:

[2]Altera DE2 SDRAM Controller `http://whoyouvotefor.info/altera_sdram.html`

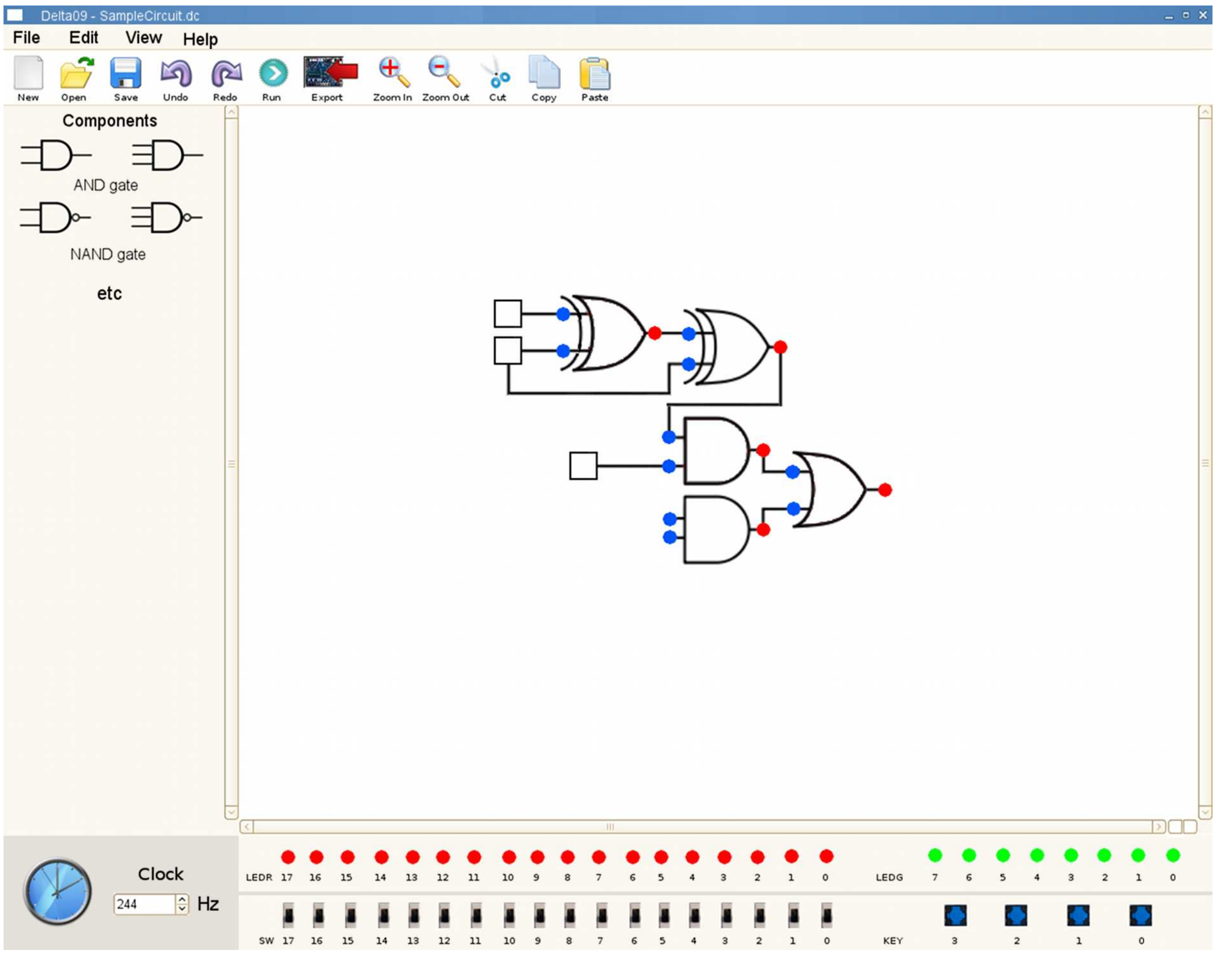

Figure 5: Mockup of graphical interface.

(*a*) A scrollable main panel where the circuit to be simulated is designed.

(*b*) A scrollable panel to the left of this, which contains a list of circuit components (gates, ROM/RAM, RS-latch, D-flip-flop, fixed inputs) with graphics and text descriptions.

(*c*) A panel underneath the main one containing diagramatic representations of all the DE2 board's toggle switches, push-buttons and red and green LEDs.

(*d*) A panel in the bottom-left corner of the window containing a clock icon and an adjustable frequency spinner.

(*e*) A toolbar at the top of the window containing:

- (*i*) New circuit.
- (*ii*) Load circuit.
- (*iii*) Save circuit.
- (*iv*) Undo.
- (*v*) Redo.

(*vi*) Transfer to board.

(*vii*) Export to Verilog.

(*viii*) Zoom in.

(*ix*) Zoom out.

(*f*) A menu bar containing the following menus:

(*i*) File

i. New.
ii. Open.
iii. Save.
iv. Exit.

(*ii*) Edit

i. Undo.
ii. Redo.
iii. Cut.
iv. Copy.
v. Paste.
vi. Delete.

(*iii*) View

i. Zoom in.
ii. Zoom out.

(*iv*) Help

i. Contents.
ii. About.

The user should be able to drag components from the left or bottom panels and drop them in the main panel to form part of the circuit. The interface will allow unlimited uses of each type of component with the exception of the LEDs, which are single-use only. Each component in the main panel will have one or more ports for input, which are represented as blue circles, and/or output, which are represented as red circles. Wires will be constructed by clicking on one of these ports, followed by clicking on a port of the opposite type. The interface will disallow multiple wires being connected to the same input port, and wires being connected from one input port to another (and similarly with output ports). The user will be able to select intermediate points for the wires by clicking as many times as desired between the source and destination ports.

The user should be able to select components and wires in the main panel by clicking on them, which will highlight them. Multiple components and wires will be selected by dragging out a box. Selections can be moved by dragging. They can also be cut, copied, pasted and deleted using the edit menu, buttons on the toolbar or keyboard shortcuts.

The user will be able to change the size and position of the display of the circuit by zooming using the toolbar/menu bar controls and scrolling.

The current circuit will be simulated on the board when the user clicks the"transfer to board" button. This will then be greyed out until a change is made to the circuit.

10 Class Design

10.1 Main UML diagram

See figure 6.

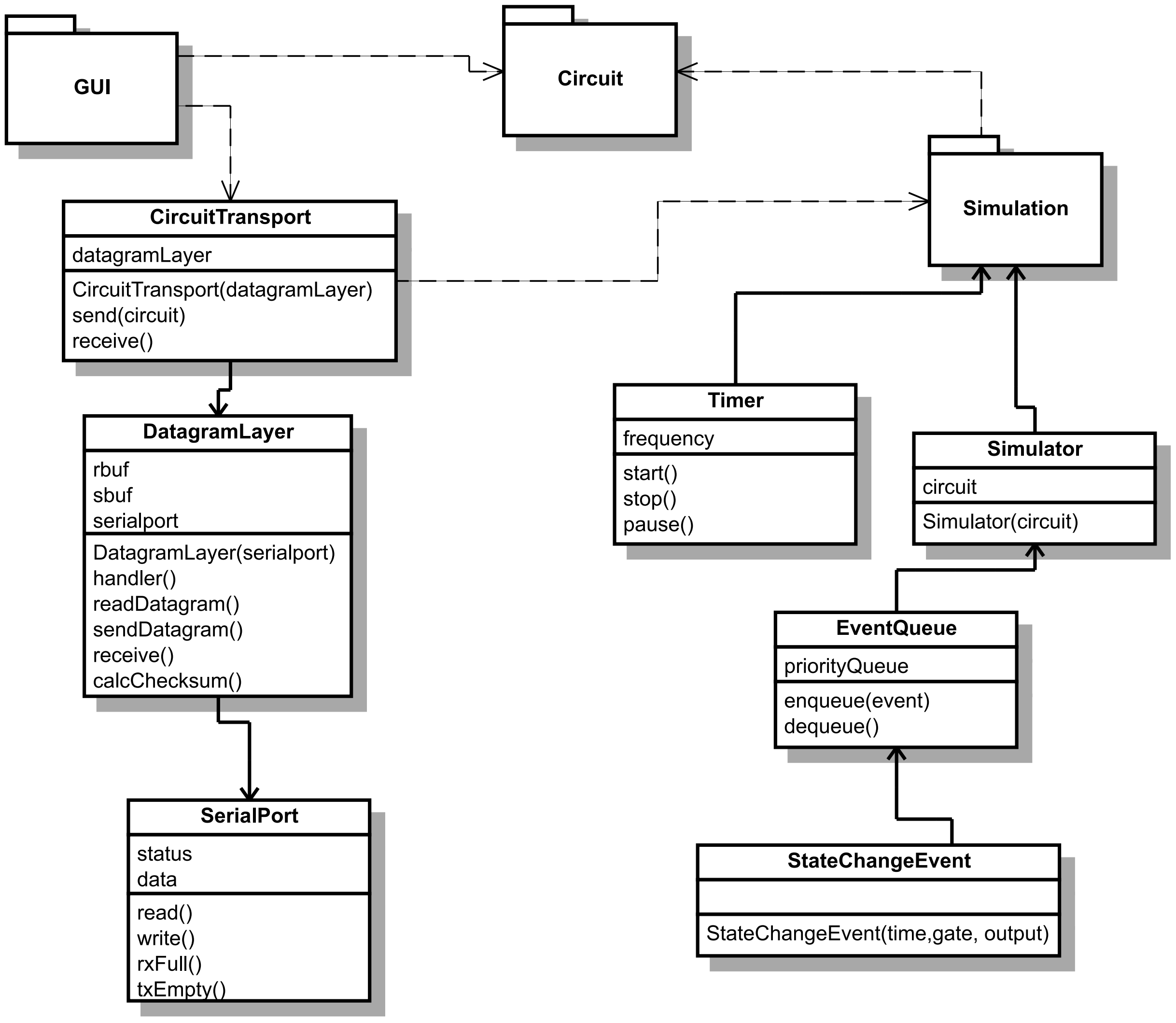

Figure 6: Main UML diagram.

10.2 Circuit data structure UML diagram

See figure 7 and 8.

10.3 GUI UML diagram

See figure 9.

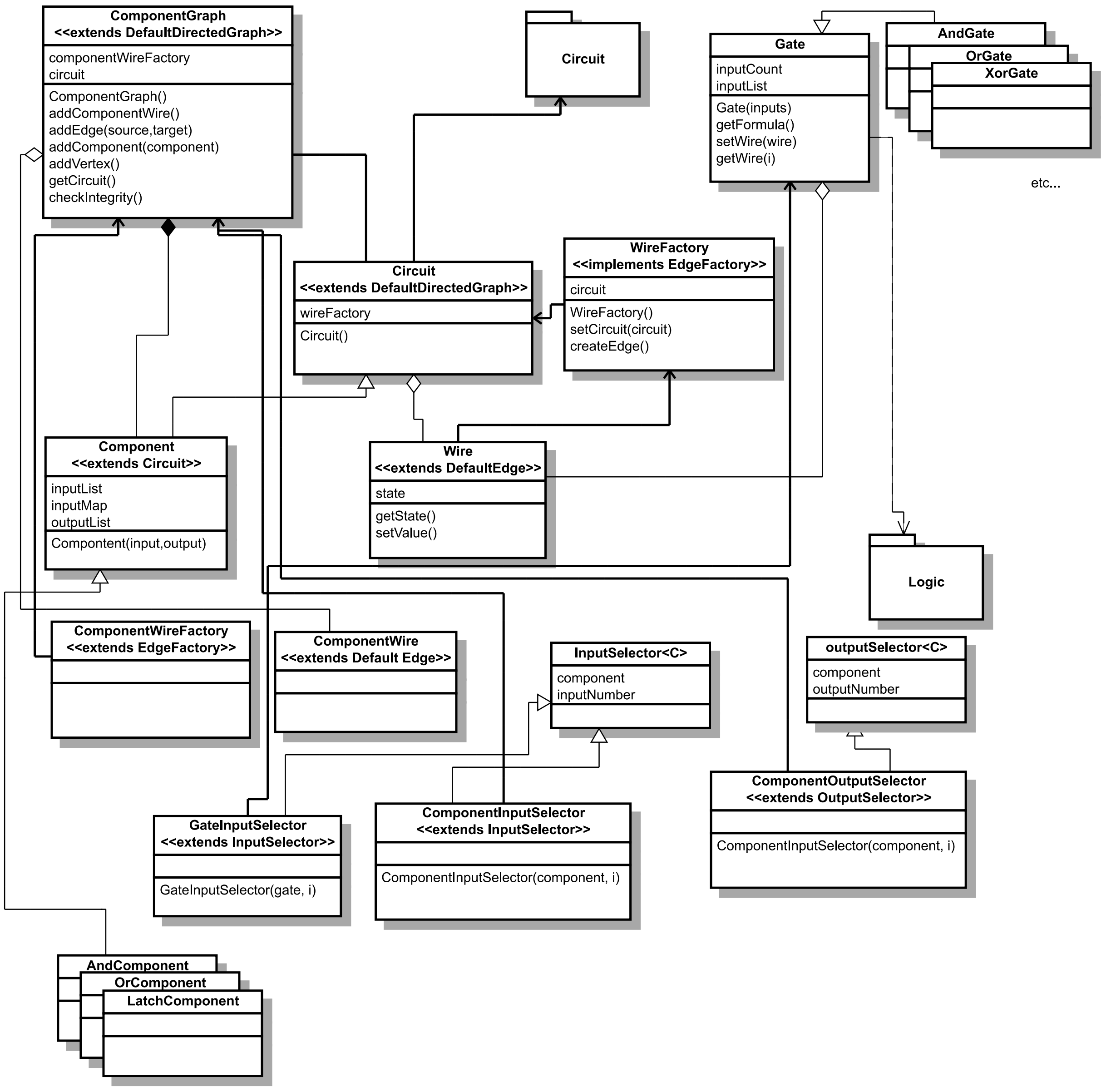

Figure 7: Circuit data structure UML diagram.

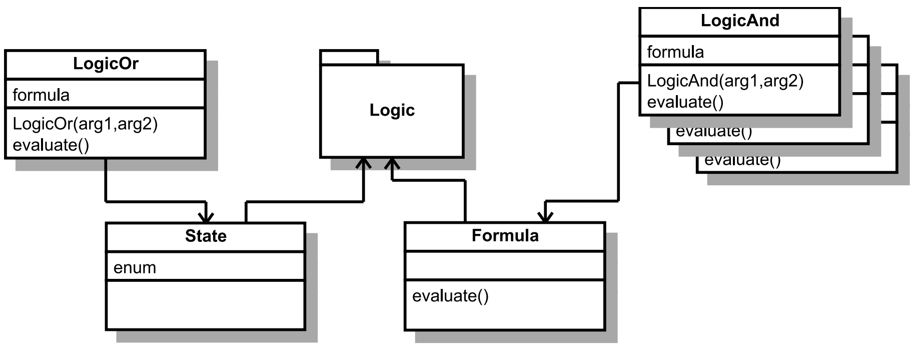

Figure 8: Logic UML diagram.

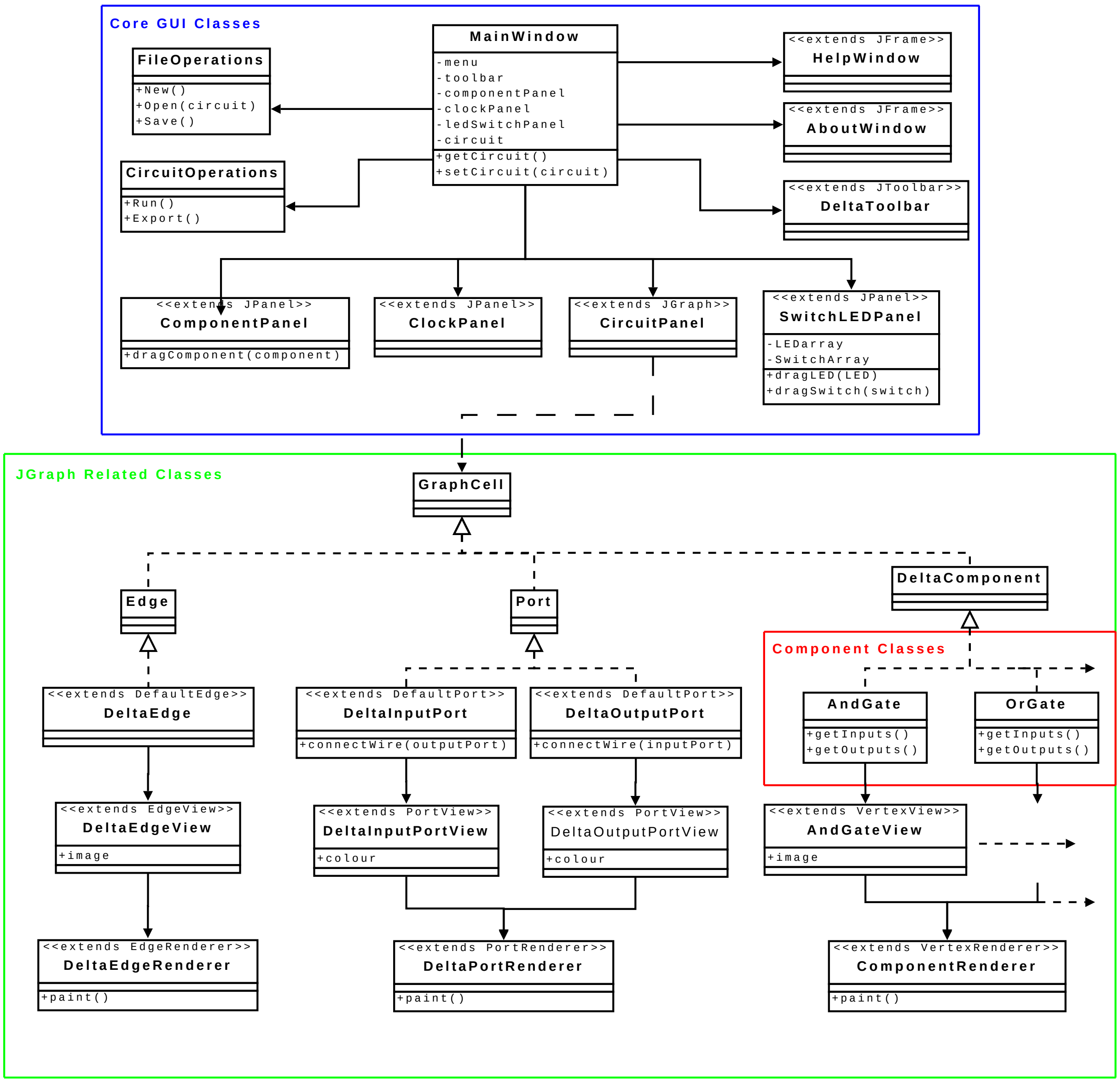

Figure 9: GUI UML diagram.

11 Distribution of responsibility

This is a rough outline of technical responsibility for the project.

Robert Duncan	Project manager, serial link & hardware-software integration.
Justus Matthiesen	Data Structure & Simulator.
David Weston	GUI – Circuit representation.
Christopher Wilson	GUI – Graphical.
Rubin Xu	Hardware – *JOP* integration, Java interfaces to board devices.

More information on the break down of task is available in the project plan.

12 Timetable

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
25th Jan	*26th Jan* Meeting at 1pm to finalise spec. and project plan.	*27th Jan*	*28th Jan* Deadline @ 12pm for specification.	*29th Jan* Client meeting @ 4pm: discuss specification and project plan.	*30th Jan* Start Coding.	*31th Jan*
1st Feb	*2nd Feb*	*3rd Feb*	*4th Feb* Group meeting.	*5th Feb*	*6th Feb* Finish major module implementation.	*7th Feb* Begin modular testing
8th Feb	*9th Feb* Group meeting.	*10th Feb*	*11th Feb* Deadline @ 12pm for progress report.	*12th Feb* Client meeting @ 4pm:discuss progress report and testing.	*13th Feb* Corrections/bug fixes. Systems testing.	*14th Feb*
15th Feb	*16th Feb*	*17th Feb*	*18th Feb* Group meeting.	*19th Feb*	*20th Feb* Final version tagged.	*21st Feb* Start writing reports.
22nd Feb	*23rd Feb* Group meeting.	*24th Feb*	*25th Feb* Deadline @ 12pm for group and individual reports	*26th Feb* Client meeting @ 4pm.	*27th Feb*	*28th Feb*
1st Mar	*2nd Mar* Code completion deadline.	*3rd Mar*	*4th Mar* Public presentation.	*5th Mar*	*6th Mar*	*7th Mar*

13 Contributions

Robert Duncan	Compilation of document, logic gates description, Verilog section, some UML diagrams.
Justus Matthiesen	Data Structures & Algorithms descriptions, including diagrams.
David Weston	Description of GUI.
Christopher Wilson	GUI Mockup, UML diagram of GUI.
Rubin Xu	Hardware description, JOP processor description, serial link description.