Project Delta
An Interactive FPGA Circuit Simulator

Individual Report
Robert Duncan

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

February 24, 2009

Contents

1 Contribution to project

I was the project manager and I also had coding responsibilities. The aim was to have a 50:50 split between these two distinct responsibilities.

1.1 Project management

I was the only volunteer for the rôle of project manager, so I was duly elected at our first meeting on the first day after the briefing lecture.

During the first week, I spent a lot of time doing a general investigation into the problem itself, and also the problem of managing the group. I decided on standardising our coding environment (where Java was used) as Eclipse, SubEclipse/command line `svn` as our interface the SVN repository and hosting our repository at *Google Code*.

I decided on using *Google Code* due to the wide variety of project management features that it includes as well as hosting SVN repositories. *Google Code* offered us:

(*a*) SVN repository for code storage.

(*b*) Full SVN log history in any easy to use format.

(*c*) Wiki pages for comments and information for the group as a whole.

(*d*) Issue tracking feature.

We also made use of email to communicate with each other outside of group meetings. I felt that email communication with some members of the team was difficult at times.

I feel that I didn't divide up the project particularly well initially, as certain members had much more work than others to begin with. Many members of the team ended up having particularly end-heavy work loads.

I took a slightly *ad hoc* approach to organising group meeting each week. On average, we did have at least one meeting a week, though I took the view that overly formalising the meeting structure was unnecessary for the project. Meetings were held as and when I, or another member of the team, felt it necessary.

I took notes in the meetings by hand, and I sent emails to the group after the meeting, when necessary, detailing their tasks for the rest of the week before the next meeting.

I found it difficult to keep the team motivated and sticking to deadlines, inevitably some code took longer than expected, but there were some major delays in producing working code. There were several delays which required dependant modules to be delayed.

Another difficultly we faced was when we had to change our approach to simulating the circuit on the board. However, I feel that we coped well with the large change required, and I feel I did well co-ordinating the modified rôles of the team members. Though thanks to the modular architecture we had designed, it didn't mean dramatically more work.

I took on the task of writing and compiling others' contribution into the formal documents needed for each of the client review meeting. This took up a considerable amount of my

time during the weekends where we had a client meeting scheduled. I tried to keep a consistent professional style within the documents produced, which I feel that I have succeeded in.

Since we were only a group of 5, I also took on some responsibilities for coding certain aspects of the project.

One of my final tasks will be to give the group presentation at the demonstration day next week.

1.2 Programming

I had two major roles within coding: one was as a general dogsbody integrating code within the GUI and providing glue methods and classes, the other was coding the verilog translator.

1.2.1 Board interface

I wrote the code that is used as the interface between the `HostDatagramLayer` that provides the low level implementation of the USB transfer protocol and the `Simulator` that produces the change events that need to be sent to the board.

The board interface is a long-running thread that is continually sending LED states to the board and retrieving switch states from packets sent from the board.

1.2.2 Translation

In our first client meeting, our client asked for translation into multiple languages to improve the user experience with the GUI. I implemented a custom `PropertyResourceBundle` class that supports UTF8 `.properties` files which is not provided in Java 6. This has meant that we have been able to include non-latin characters easily into the translation, allowing us to offer English, French, Arabic, German, Chinese, and Japanese. I wrote the French translation. I also integrated the translations into the GUI so previously built-in strings where abstracted away to the `Translator` class.

The main difference between the standard `PropertyResourceBundle` and my custom `UTF8PropertyResourceBundle` is in the file handling code:

```
@Override
protected Object handleGetObject(String key) {
   String value = (String)bundle.handleGetObject(key);
   if(value == null) return null;
   try {
      return new String(value.getBytes("ISO-8859-1"),"UTF-8");
   }
   catch(UnsupportedEncodingException e) {
      return null;
   }
}
```

I also modified the layout code in `MainWindow` to allow for right-to-left languages both in menus and text, but also the main layout, so that the side panel is flipped in Arabic mode.

1.2.3 GUI tasks

I implemented all of the Mnemonics for all the menu items, again using the `Translator` class to enable multi-lingual mnemonics.

I designed the SVG icons for the majority of the components. I also added most of the components to the graphical interface using the `ComponentPanel` and the `CircuitPanel`.

I wrote the `Export`, `Run`, and `Stop` GUI actions to control the simulation and to export the circuit into Verilog.

Adding *JavaHelp* code to display help file, and designed the About dialog box.

1.2.4 Verilog translator

I also worked on translating between the `ComponentGraph` representation of the circuit and a verilog interpretation. This was made more difficult by the continuously updating method names and functionality added to the Gates and Components as part of the work on the Simulator. This has meant that `VerilogConverter` has had to have been rewritten (at least in part) several times to reflect changes in the data structure.

I had to include file handling code within `VerilogConverter` to allow for inserting the generated Verilog code into a *Quartus* project file and then copying the entire directory structure to the save location specified by the user in the GUI.

I took a modular approach to the translation, with a translation phase that operates on the `ComponentGraph` layer initially, connection the wires appropriately, then it hands off the `Component` to translate itself into Verilog, either by defining a custom declaration, or by calling a static method in `VerilogConverter` that works on translating a `Circuit` (the underlying simulatable component representation) into Verilog. The method that works on the circuit performs a similar set of steps to the previous method, but this time it has to take into account the difference between `ComponentWire`s and `Wire`s which were internal to the component.

I had to take extra care when components have more than one output, since the simplistic wire routing I had used wouldn't have taken the additional wires into account. I resolved this issue by adding extra `assign` statements to the verilog translation of the `Gate`.

For example, this is how I dealt with output gates from a `Circuit` inside a `Component`:

```
if(circuit.outgoingEdgesOf(g).size() != 0) {
   for(Wire w : circuit.outgoingEdgesOf(g)) {
      output.add(wireNames.get(w));
   }
}
else {
```

```
   //i.e. we are an output gate from this circuit
   for(int i = 0; i < component.getOutputCount(); i++) {
      for(ComponentWire w : component.getOutputWires(i)) {
         output.add(outputWires.get(w));
      }
   }
}
```

2 Evidence of contribution

2.1 Project management

The *Google Code* page for our project can be found at
`http://code.google.com/p/delta09/`

(*a*) Specification (with group contributions)

(*b*) Progress Report (with group contributions)

(*c*) Final Report (with group contributions)

Meeting minutes were taken by hand, so are not included here. Emails were sent our after the meetings with task to complete for the following week.

I will also be presenting our group presentation on the demonstration day next week.

2.2 Programming

I will not include code here, but just reference classes where I have contributed.

(*a*) `org.delta.verilog.VerilogConverter` – sole reponsibility.

(*b*) `org.delta.transport.*` – provided most functionaility in both classes.

(*c*) `org.delta.gui.i18n.*` – provided all Java code, French translation.

(*d*) `org.delta.gui.{Run, Stop, Export, Help, About}Action` – provided `actionPerformed(ActionEvent e)` method for each.

(*e*) `org.delta.gui.diagram.{RAM, ROM, RSLatch, SevenSegment, Switch, DFlip, Nand3Gate, Nor3Gate}` – Created these classes to integrate components into the GUI.

(*f*) `org.delta.gui.MainWindow` – provided functionality for `Translator`, added mnemonics, added Settings file.

(*g*) `org.delta.circuit.component.*` – Added `getVerilogMethod()` to each component.

(*h*) `org.delta.circuit.gate.*` – Added `getVerilogMethod()` to each gate.

3 Contribution of other group members

3.1 Justus Matthiesen

Justus spent most of his time working the data structure used for simulating the circuit and on the simulator itself. He produced some clean, well-abstracted code. He was accessible via email, and receptive to interface requests and bug fixes. He contributed well in meetings.

3.2 David Weston

David spent most of his time working on the GUI. David took a while to get into the flow of using the SVN repository to check in his code. I felt that he was, at times, difficult to contact via email. As the weeks progressed David took on more work and has produced an attractive GUI experience to the user.

3.3 Christopher Wilson

Chris spent most of his time work with the JGraph library to develop the visual aspect of circuit creation. He also worked with Justus to integrate the visual data structure with the abstracted data structure used to simulate the circuit. Chris was easy to contact and has put a lot of effort into this project.

3.4 Rubin Xu

Most of Rubin's work was in the first couple of weeks when he was developing the JOP interface and working with the hardware. Rubin has produced a very efficient interface between the board and the PC which has enabled to circuit to be simulated in Java on the PC. Rubin seemed to put a lot of effort into producing good abstractions for the rest of the team to use when interfacing with the board.