

A JAVA BREAD-BOARD SIMULATOR: DIGITAL CIRCUIT SIMULATION WITH AN OPEN-SOURCE TOOLSET

Chris Bailey, *Department of Computer Science, University of York, Heslington, York, UK*

Michael J Freeman, *Department of Computer Science, University of York, Heslington, York, UK*

ABSTRACT

Digital electronics is an area of student learning that benefits substantially from ‘hands-on’ experience. Simply simulating circuits at a high level will not instill a full understanding of the pitfalls involved in circuit building, testing and design in the real world. Consequently most electronics related curriculums will include practical lab-work to supplement any other activities to be delivered as part of a course module. At many UK universities the use of ‘bread-boards’ is common. These are rapid circuit construction boards, which allow circuits based upon chips to be wired and tested. However, it is less practical for students to undertake such work unsupervised (due to health and safety legislation), and also often not practical for them to undertake this work at home. Consequently, a Digital Bread Board simulator to supplement such teaching styles is a valuable teaching aid. This paper describes the Bread-Board Simulator developed at the University of York over a number of years, and a new project to release the tool-set as an Open-Source Learning Platform.

KEYWORDS

Digital Simulation, Bread-Board, Virtual Learning Environments, VHDL Simulation, Open-Source

1. INTRODUCTION

The use of bread-board circuit construction is a common methodology for practical digital electronics in UK universities. A Bread-Board is a circuit module that can have chips and wires inserted and removed without permanent connection, and thereby allows rapid construction and modification of test circuits. As such it presents an ideal medium for teaching

of digital circuit design. However it is less convenient for students to make use of this medium at home or out of lab hours, and therefore students are limited to the amount of extra work they can do if seeking to learn at their own pace. This is especially so where increasing pressure to maximize utilization of lab facilities makes freelance access to facilities restrictive. Consequently, a learning tool that can be used on a computer platform as a supplementary learning tool, mirroring the functionality of a bread-board environment, was perceived to be a valuable goal by the first author. Subsequently, he established a series of student dissertation projects to develop the idea as an extensible platform for digital circuit experimentation – the Java Digital Bread-Board (JBB) was thus developed, and is described here in this publication.

Please note that this paper describes in the authors own words, work undertaken over number of years by final year dissertation students under his direction, as per the acknowledgement at the end of this paper. It then discusses the further extension and future open-source development of the platform under a new project initiative. JBB was first evaluated in the EU Funded NETPRO Projects [Donazelli 1999, Bailey 2002], and has continued to be developed since.

2. JAVA BREAD-BOARD

A ‘Bread-Board’ consists of a module (several of which may be interconnected to create multiples of larger size), upon which a series of interconnects are provided. Chips may be placed in the boards by inserting chip pins into the pin sockets, and extra wires may be added to complete a circuit. Typically, test equipment such as an oscilloscope can be used to monitor real-time circuit behavior, or Light-Emitting Diodes (LED’s) can be used to monitor circuit signals for slower circuit operations. Figure-1 shows an example of a bread-board. Standard components that are used with bread-boards include TTL logic chips, a range of standard logic functions, push-switches, for generating test inputs, and light-emitting-diodes (LED’s) for displaying signal states on outputs.

Since the range of TTL logic chips is clearly defined, as are their circuit characteristics, then provision of such chips in a simulation environment is a relatively well-defined task. However the possibility of new chips being introduced, or of a more complex or custom component being required in a learning exercise makes it highly desirable for an extensible set of components to be available. In a laboratory this is easily achieved by simply sourcing an appropriate component, or programming a configurable device to behave in accordance with the requirements.

However for a simulation environment this is less straightforward. It is undesirable to have to rebuild the application for each new device being added to the chips available, and especially so when a new chip is being incrementally tested and developed, either by the tutor or a student. Consequently the objective of the Java Bread-Board project was not only to provide a convincing and familiar bread-board environment in virtual form, but provide relatively easy methods of adding new chips to allow extension of the system beyond its original intentions. This was the basis of our project and will be explored further in this paper. However it is best to first introduce the JBB environment and its capabilities before talking further about the extensibility and the more advanced features that this has allowed to be developed as part of the JBB tool-set.

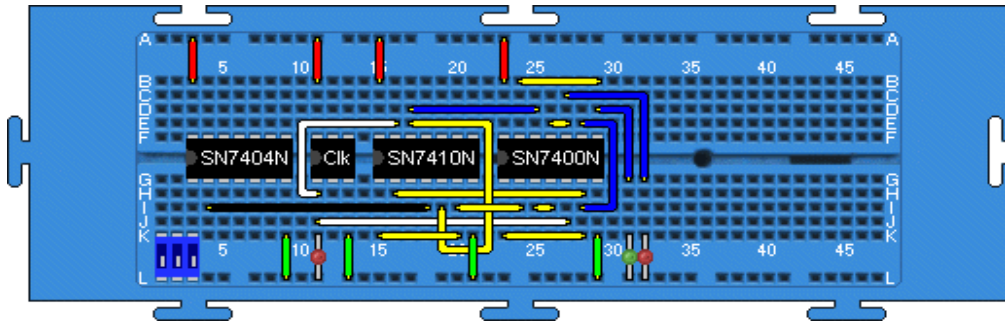


Figure 1. Example of a Bread-Board Circuit

3. JBB CORE APPLICATION

Figure-2 shows the Java Bread-Board (JBB) application in use. A Tool-Bar provides access to a number of functions, whilst a left-hand panel gives details of the currently selected component. The right-hand (main) panel shows the bread-board circuit, and can scroll to accommodate much larger circuits. Circuits can of course be loaded and saved, making it easy to develop a circuit over a period of several sessions, share and distribute examples, and perhaps in some cases, provide an existing circuit for a student to experiment with (perhaps verify its operation for example, or identify a design fault and remedy it).

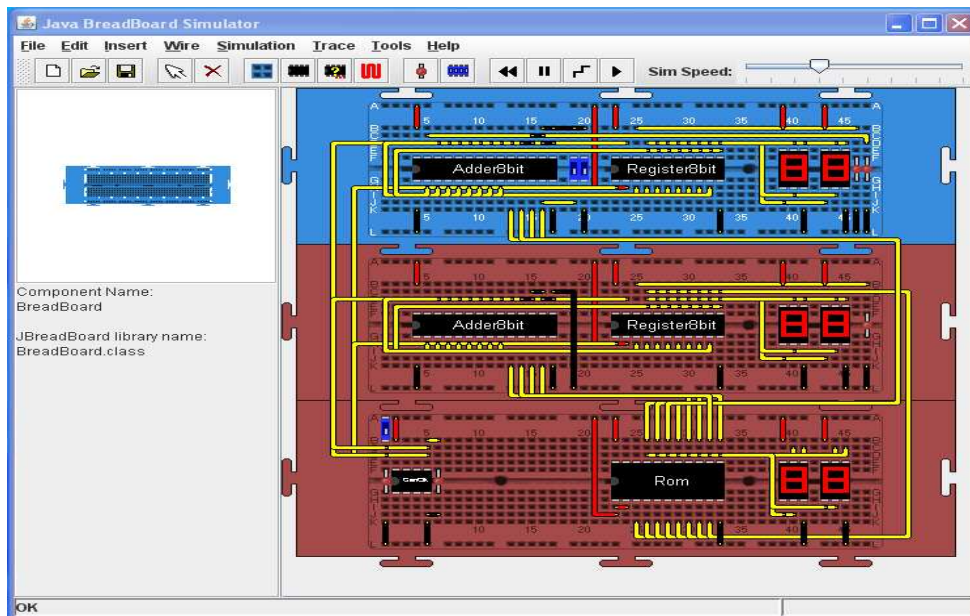


Figure 2. JBB Application in use (MS Windows)

Once a circuit is considered complete, it can be simulated by step or multi-step simulation, at various speeds, with circuit chips behaving with the correct circuit delays for their given design and specification. Consequently it is possible to see effects such as propagation of signals through successive chips, for instance the ripple-through effect in a counter, and also to observe logic glitches as they would be seen in a real circuit. This is important, as a simply functional behavior would not be sufficient to allow a true representation of a circuit that could be built in a practical lab session and behave the same way as the bread-board virtual circuit model.

Wiring can be color coded, which is not only useful in aiding circuit understanding, but a good practice to be employed in real circuit construction. Similarly there is a choice of several LED colors (RED, GREEN, YELLOW) to permit output values to be grouped. For example the outputs of a 4-bit adder may be wired to a red LED for the 'Carry-Out' signal, whilst four green LED's might be used for the four output sum bits.

Probes and Traces are also supported, by means of special 'probe' components, which can be placed on any circuit contact (these appear as small squares with a 'P' label). Multiple probes can be placed at different circuit locations, and during simulation, these probe values are dumped to a trace file. The examination of the trace file provides a time-stamped logic trace of circuit behavior at each simulation step. This can be examined directly, or a tool could be developed to visualize the trace outputs as waveforms, such as a virtual logic analyzer or oscilloscope applet.

4. THE VALUE OF EXTENSIBLE AND OPEN-SOURCE DESIGN

A major design decision, made at the beginning of the JBB project, was to develop the platform to be as open as possible to future developments. Although this was not initially achieved to 100% satisfaction, the ability to add new chips and other components to the JBB library is an important feature of the system. Later it was realized that the ability to interact with some components by double-clicking them would have been useful, and the core application code was then modified to allow this [Page 2004]. Subsequently there have been a number of developments that make JBB more useful, especially for teaching and learning purposes. This is a feature that we expect to further exploit in new developments (see section 4.5).

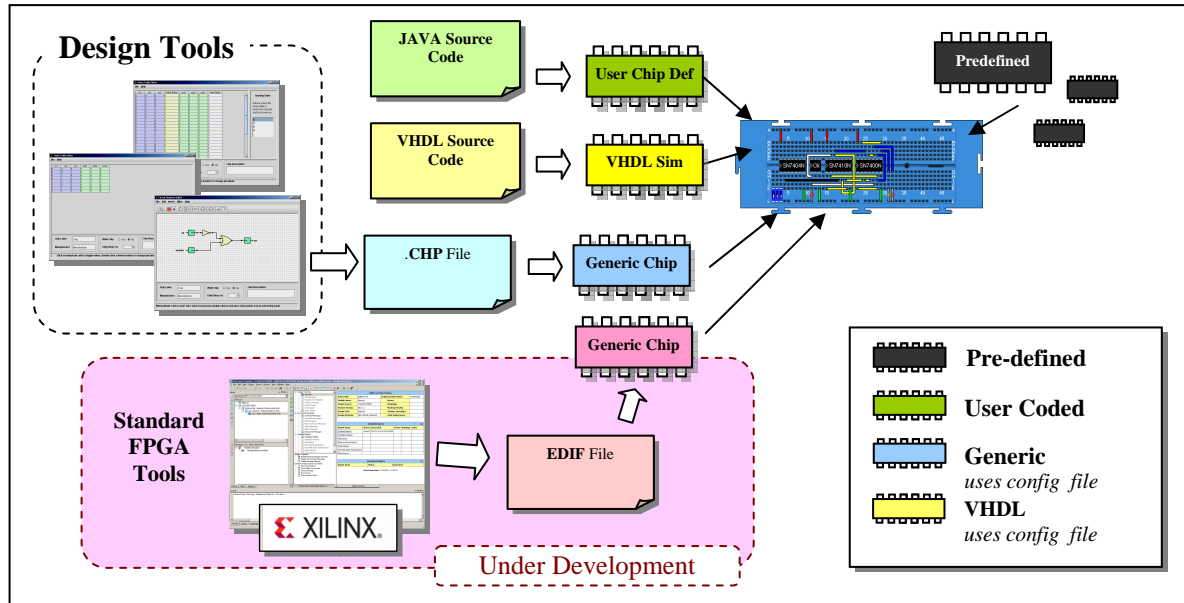


Figure 3. Chip Creation Routes

4.1 Creating New Chips in Java

The primary extension feature of JBB is the ability to add unlimited Java class files into the chip directory. These class files must follow a template for a chip, but within that code wrapper any Java code can be incorporated. This means that components can be modeled and implemented in Java as simple logic gates, or something as complex as a complete microprocessor or microcontroller. This feature is of great assistance for the JBB development team themselves and for those, such as tutors, who can devote some resources and ingenuity to developing a new chip for a given purpose. However students may find this a less friendly way to develop new chip functions.

4.2 Simple Chip Design Tools

Whilst a Java chip creation methodology provides unrestricted scope for chip creation, the average student may wish to do something at a simpler level, without specialist knowledge. Additionally the ability to rapidly modify and re-test a student-defined chip is valuable. In order to facilitate this pedagogical need, a new project was initiated as an extension to the JBB platform, and resulted in the development of three chip design tools [Halstead 2005]. These include a table-based designer, a schematic capture tool, and a state-table entry tool. (see Figures 4, 5 and 6 for screenshots). In each case a design can be created and translated into a chip description file, used by the 'Generic' chip module, which simulates the design behavior. Figure-3 shows the various methods for defining or coding chip behavior, including our latest project for EDIF files.

4.3 Advanced Chip Design Tools

The requirements for table-based and schematic-capture design methodologies are provided for by the JBB partner tools as described in section 4.2. However a more recent and increasingly dominant design methodology is that of the Hardware Description Language or HDL. The most popular being Verilog and VHDL. The JBB system now has an experimental module extension that permits simulation of VHDL models for chips, allowing a student to design a chip using VHDL source text [Buse 2008]. This places the JBB capabilities closer to the FPGA-VHDL design methodology likely to become predominant in teaching electronics in the curriculum of most universities. Further work, based upon importing EDIF files from FPGA tool-sets is currently ongoing [Pfister 2010].

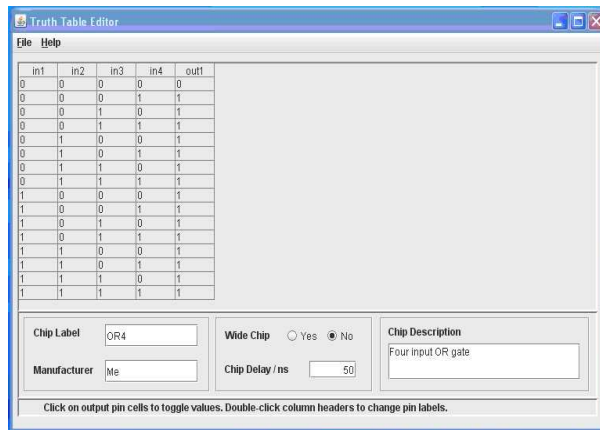


Figure 4. Truth Table Design Entry

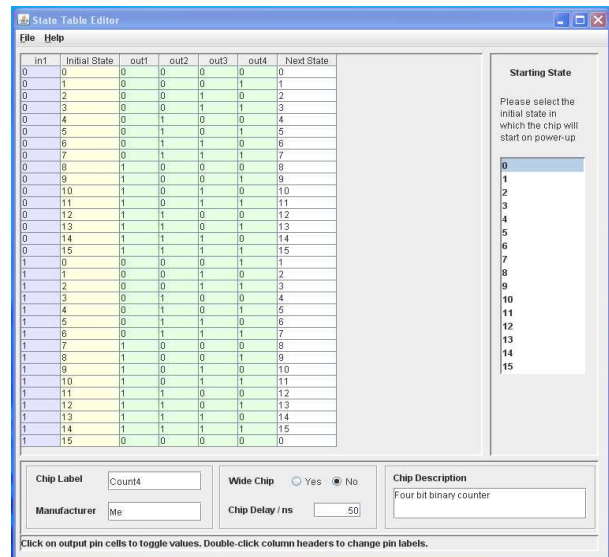


Figure 5. State-Table Design entry

A JAVA BREAD-BOARD SIMULATOR: DIGITAL CIRCUIT SIMULATION WITH AN OPEN-SOURCE TOOLSET

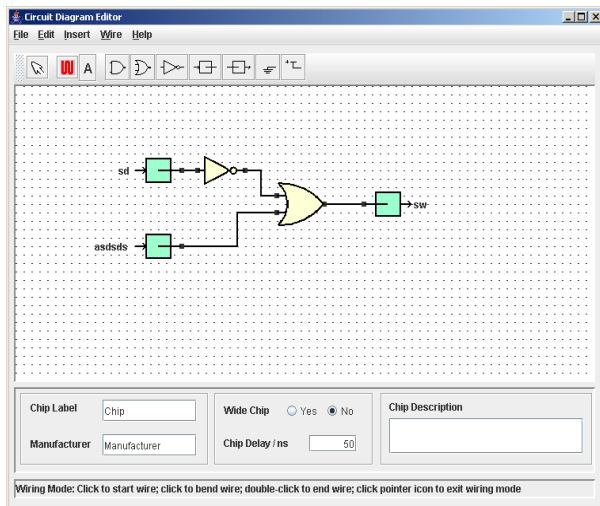


Figure 6. Schematic Capture Design Entry

```
##JBB NAME "mem" DESC "mem IC"
MANU "djb1312" PINS 16 ;
```

```
##JBB PINMAP 0=A(2) 1=A(1) 2=A(0)
8=clk 3=O(2) 4=O(1) 5=O(0) 12=D(2)
11=D(1) 10=D(0)END PINMAP;
```

entity mem **is**

```
generic(width: integer:=7;
        MemItems: integer:=50000);
```

```
port( A: in integer;
      D: in bit_vector(width-1 downto 0);
      O: out bit_vector(width-1 downto 0);
      clk: in bit
    );
```

```
type datatype is array(MemItems downto 0)
of bit_vector(width-1 downto 0);
```

```
variable data: datatype;
```

```
end mem;
```

architecture behv **of** mem **is**
begin

```
    O <= data(A);
```

```
    process (clk)
    begin
```

```
        data(A) := D;
```

```
    end process;
```

```
end behv;
```

Figure 7. VHDL Example (Buse2008)

In order for a VHDL description to be used for a chip, a specialized chip was developed in Java which interprets VHDL code from a source file, and generates the expected pin behavior. This includes any internal storage, such as memory arrays, which means that a VHDL model of something as complex as a microcontroller could be modeled on the JBB. The VHDL interpreter has been built to be further developed. Its parser structure means that a full and complete VHDL syntax could eventually be provided. However its current capabilities are still good, the VHDL code in Figure-7 illustrates this: the simulator can deal with generic parameters and arrays, making quite a range of possible designs compatible [Buse 2008]. Figure-7 also shows the first lines of the VHDL file including a special chip pin-map section which is used to allow the VHDL to be configured as a bread-board chip item.

4.4 Test and I/O Functions

Features not yet described in this paper include some specialized chip and component modules which are worthy of mention. In particular there are 7-Segment display modules, and a hex-keypad 'chip' which allows the user to click and bring up a key-pad for data entry (see Figure-8), as developed by Rob Page [Page 2004]. This is analogous to the sort of basic input facility likely to be found in a lab. Simpler dip-switch inputs are however provided for a more traditional approach. Memory chips (as used in Figure-2) are also available.

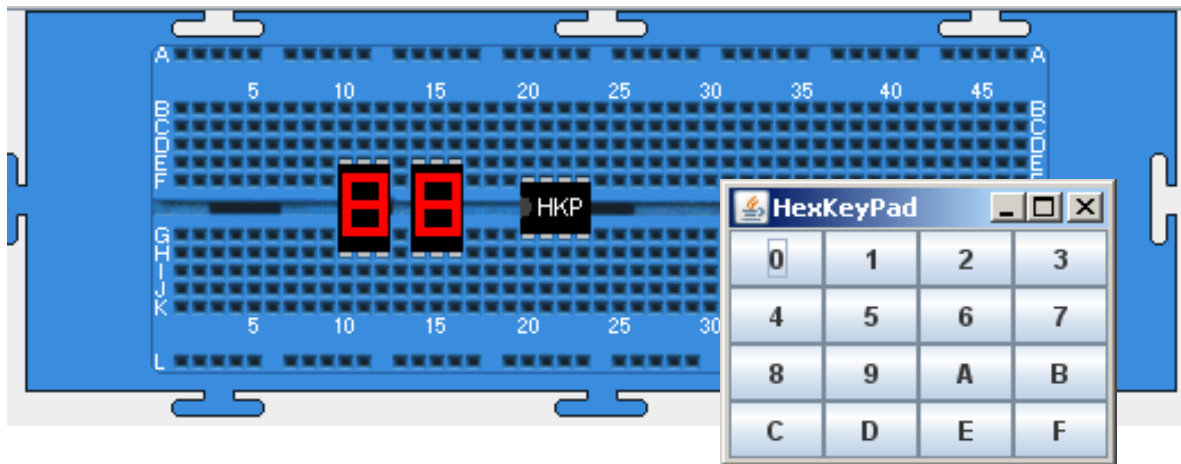


Figure 8. Seven-Segment Displays and Hex Keypad with Header chip

4.5 Future Extensions

The Scope for extensions to JBB is very wide, and only limited to the developer's own ingenuity. Particular areas of interest do however come to mind. First of all, the ability to emulate a small FPGA component from a manufacturer's standard device family would be extremely useful. This would allow a standard design toolset such as Xilinx Web-PackTM to be used to generate circuit models that could then be 'loaded' into the FPGA emulator chip. To

date the experimental work on developing test and measurement virtual equipment has not been fully satisfactory. However this work at least confirms it should be feasible to develop new modules with GUI interfaces that mimic test equipment such as logic probes, oscilloscopes, and logic analyzers. Examples of other developments that are envisaged for future development include:-

1. **Emulation of FPGA Netlist output at component level (e.g. CLBs)** *-under development*
2. **DSP workbench using chips as DSP sub-modules**
3. **Standard CPU Emulation via JBB chips**
4. **CPU WorkBench using chips as internal simplified CPU components** *-under development*

Some of these intentions are driven by current programme developments at the University of York. The CPU work-bench, for example, is envisaged to support first-year computer architecture delivery. This has been taught previously through the ‘paper’ evolution of an imaginary machine from a basic register and adder through to a full CPU design. Having a simulation environment that can operate at the level of block-components would be very desirable, as it allows a hands-on low-level appreciation to be retained in the lab-style, whilst considering relatively abstract levels of machine architecture in lecture delivery.

The DSP workbench would allow individual chips to communicate via single-wire serial data (e.g. 16 bits per input source). This would be utilized as Java Bread-Board does not support analogue signals. The ability to link DSP units together via single wire inputs and outputs would permit an abstract level of DSP formulation. Modules might include filters, oscillators, mixers, and so-on, providing an audio or radio-frequency context for experimentation with techniques such as modulation and band-filtering.

Emulation of FPGAs (Field Programmable Gate-Arrays) will be possible by taking output from industry standard tools (of which many are available free to all types of users), and simulating the behavior of the design components known as ‘CLB’s (configurable Logic Blocks). Consequently a chip’s behavior can be defined using industry standard tools, such as XILINX Web-PackTM and even potentially make use of standard design libraries that may pre-exist in the community. Provided that a tool can generate an EDIF netlist, then VHDL, VERILOG, possibly even Handel-C and other Hardware Description Languages could be used in the future. This particular extension is under development at present [Pfister2010] , a screen-shot of an initial EDIF ASIC Chip being used in the bread-board is shown in Figure-9.

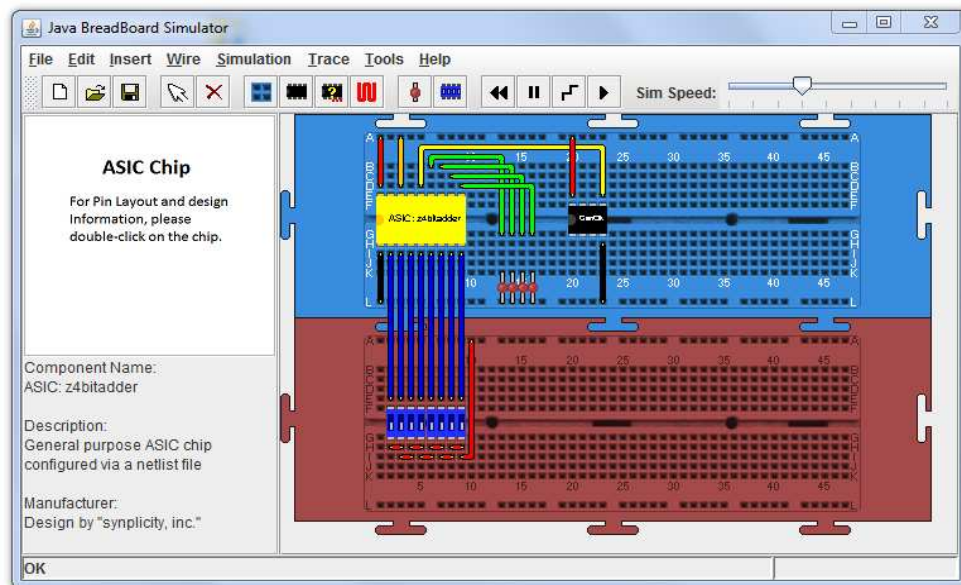


Figure 9. 'ASIC' Chip Example

Whatever extensions may be developed, the important point to note here is that any of these suggested new developments would be achievable simply by independent development of new chips. JBB will recognize these automatically and the software suite does not need to be rebuilt each time a new chip is added. Therefore from the point of view of software development, the individuals undertaking the work require only a very limited knowledge of the JBB internal functionality at the chip-level interface.

During our current HEA/JISC funded open-source initiative the Chip-Interface class has been completely revised, as compared to the first JBB toolset development releases. The new Interface class (see Figure-10) allows a more flexible approach to chip creation, allowing any chip to be augmented by derivatives simply by adding a new sub-class. For example, a TTL chip 7400 may be supplied as a Generic type, but we may wish to add derivatives such as 74LS00, 74AC00, and so-on, as illustrated by an example in Figure-11.

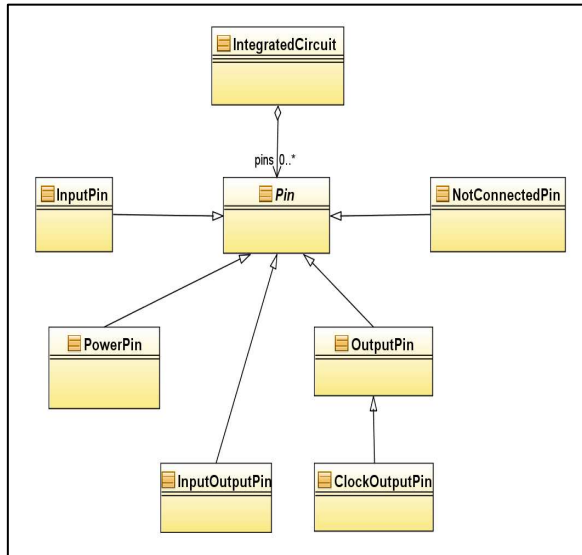


Figure 10. 'IntegratedCircuit' Interface Class

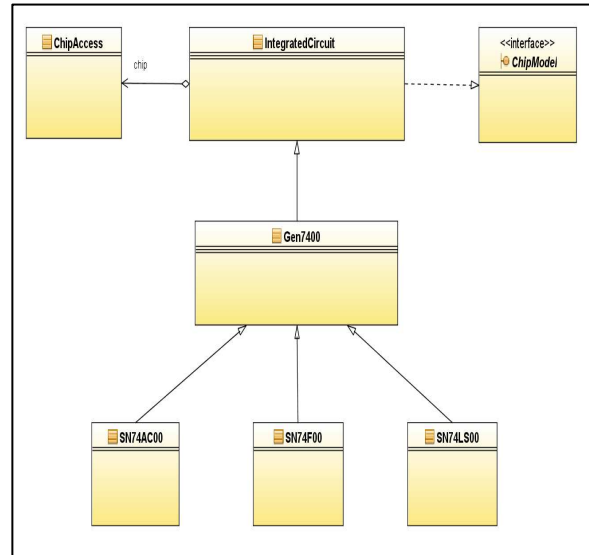


Figure 11. Example of Chip derivative Extension

Further extensions to JBB Tools are also considered worthwhile, but require a more controlled open-source development approach. There are many improvements that could be made to JBB simply for current use as a digital circuit simulator. Additional extensions such as a logic analyzer for instance have been attempted in previous student projects. The level of work required to meet a supportable end-product release does however require substantially more work, something we hope will be supported by open-source communities with an interest in electronics teaching and learning. Again, extension without rebuilding the full software (ie a plug-in methodology) is being developed, such that a community of users can obtain individual features and install them in their own local software setup interchangeably. Tool developers can independently develop modules without needing to co-ordinate with each other (since the main application is untouched). Naming conventions for chips and tools are given in the user guides to prevent naming conflicts. This model of development is extremely important if an independent community of JBB contributors is to be facilitated, as envisaged by the open source initiative being undertaken now.

5. OPEN-SOURCE INITIATIVE

Currently, new funded work is being completed at York under the HEFCE/JISC Open Educational Resources Pilot Initiative¹. The OER programme seeks to develop existing learning resources for open distribution via the JORUM learning repository². The Java Bread-Board project is being placed in this repository as part of the initiative, allowing a collective learning resource library to be developed by contributors (for example learning tutorials,

¹ OER : www.heacademy.ac.uk/ourwork/learning/opencontent

² Jorum repository: www.jorum.ac.uk

exercises, and mini-projects). We also intend to develop an open source repository for the software itself – allowing significant community involvement in future JBB extensions and developments. The JBB team is currently considering use of SourceForge³ and GPL licensing as a medium to fulfill this aim. This has included some rationalization of the tool set and its internal structure, such as plug-in integration of design tools with the JBB main menus (in fact the facility to call up any supplementary tool placed in the ‘Tools’ directory now exists). The new ‘ChipInterface’ class and a ‘Design Tool’ interface class now allows third parties to independently develop both chip content and tool extensions without needing to rebuild the rest of the Java Bread-Board toolset. This is important if JBB is to be used widely by a community with varied levels of knowledge, as new developments can be supplied simply as ‘plug-ins’.

6. CONCLUSIONS

Java Bread-Board is an open-ended software platform, which we now hope will become an open-source tool-set and open-learning resource. This requires involvement of learners and teachers to be successful, and effort from the open-source development community. The potential of JBB is far larger than has been realized to date, with a single familiar environment it should be possible to cover a wide range of electronics and computer-architecture related topics in a learning context.

Perhaps the greatest success of JBB is the fact that it was students themselves who have developed this resource, creating a path for open development that we hope to continue to follow in future projects, including the Current OER funded project, and beyond.

ACKNOWLEDGEMENTS

As mentioned earlier, the JBB project was conceived by the author, and undertaken under his direction by project students in the University of York Computer Science Department, as a series of final year dissertation projects. The Author therefore gratefully acknowledges the work undertaken in realizing the JBB tool-Set, and in particular by the following former York University Students: Nicholas Glass, Shaun Gilbert, Rob Page, Stephen Halstead, and Darren Buse. The latest student contributions for the FPGA netlist emulator chip are being carried out by Kevin Pfister at the time of writing.

The recent open-source revisions and developments were funded under the HEA/JISC/HEFCE Open Educational Resources Programme (OER). The original Java Bread-Board tools are still available at www.cs.york.ac.uk/netpro/bboard whilst a new site is under development at www.cs.york.ac.uk/jbb at the time of writing.

³ Source Forge url :- www.sourceforge.net

REFERENCES

Conference paper or contributed volume

Donazelli et al 1999, Learning Electronic Systems Design with a Project Based Course on the Network , *Proceedings of the 1999 ENABLE Conference*, Helsinki, pages 114-121, ISBN 951-647-001-7, Evitech Digital Press

Bailey 2002, Enabling Network Based Learning , *Proceedings of the 13th EAIEE 2002*, York, ISBN 1-85911-009-6

Other

Glass 2002, Java Digital BreadBoard Simulator: A Simulator for an educational electronics environment, Final year project report, University of York, Dept. of Computer Science, 2002

Page 2004, Extending and Upgrading the Java Breadboard Simulator Tool, Final Year Report, University of York, Dept. of Computer Science, 2005

Halstead 2005, Circuit Design Tools for a Breadboard Simulator, Final year project report, University of York 2005

Buse 2008, VHDL Simulator Tool for the Java BreadBoard, Darren Buse, University of York 2005

Pfiister 2010, Final year project report, University of York, Dept. of Computer Science, March 2010