CircuitBoard: Sketch-Based Circuit Design and Analysis

Shane W. Zamora
University of California, Santa Barbara
Department of Computer Science
Santa Barbara, CA, 93107
char42@cs.ucsb.edu

Eyrún A. Eyjólfsdóttir
University of California, Santa Barbara
Department of Computer Science
Santa Barbara, CA, 93107
eyrun@umail.ucsb.edu

ABSTRACT

Digital logic circuit design is an inherently computable process that can greatly benefit from real-time feedback and evaluation. This paper presents CircuitBoard, an application for designing and testing hand drawn digital logic circuits using a sketch-based interface, such as on a Tablet PC. Our system aims to provide these tools to a paper-like interface, an environment shown to be natural and conducive to design conceptualization. We outline our low-level strategies for stroke segmentation and logic element recognition, as well as our high-level approaches to circuit evaluation and the motivations behind our system's user interface design.

Author Keywords

Sketch recognition, circuit design, user interfaces.

ACM Classification Keywords

H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces; J.6 [Computer-aided engineering]: Computer-aided design; I.4.8 [Image Processing and Computer Vision]: Scene Analysis. - Object recognition;

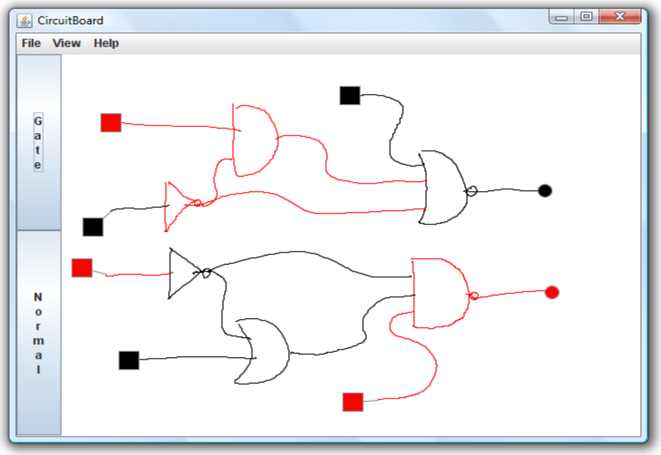

Figure 1. A digital logic circuit design hand drawn in CircuitBoard. Simulation mode is active, and the circuit is being simulated. Tapping the square figures on the left will toggle the inputs of the circuit, providing an intuitive interface for interaction. The boolean value of each gate and wire is indicated by its color – red is *true* and black is *false*. The final outputs of the circuit and their values are indicated by the circular figures on the right.

INTRODUCTION

While high-tech computer-aided design (CAD) applications are critical tools in the development of any complex digital logic system, industry professionals, researchers, and students alike prefer to begin their design process on low-tech devices such as a whiteboard or a sheet of paper. The simplicity of paper-based systems provides a more suitable environment for brainstorming and drafting digital logic circuits when compared to computer-based environments. Handwriting a prototype allows the user to quickly and intuitively externalize their thought process where the complexities of CAD applications ultimately constrain the ability of the designer to express their ideas. Whiteboards also provide an environment conducive for multi-user collaboration on the logic design and brainstorming process.

Although a hand drawn approach is well suited for drafting digital logic circuits, CAD applications can provide vital tools for automating the visualization, simulation, and debugging process. As debugging a circuit drawn on a whiteboard or sheet of paper can take intensive investigation and multiple cycles of trial and error, a CAD application that can provide tools to assist this process can be invaluable.

In this paper we present CircuitBoard, a sketch-based application for designing digital logic circuits that aims to combine the freedom of whiteboard-like interfaces and the debugging tools provided by modern CAD development environments. CircuitBoard uses a modeless pen-based interface for composing hand drawn logic gates and wires in arbitrary configurations, scales, and orientations. Although mouse input is supported, CircuitBoard is designed for the stylus-based input of Tablet PCs in order to capture the feel and benefits of a traditional pen-and-paper-based interface as described. Our system presents an intuitive and minimal interface for editing and simulating these circuits and provides the user with valuable tools previously accessible only by meticulous translation of a design into a CAD environment.

In the following sections we will discuss our implementation of CircuitBoard. We will present and explain the motivations behind our user interface design, and describe our system's implementation. We will focus on its three major components: stroke segmentation, symbol recognition, and circuit evaluation. The difficulties encountered and the robustness of our solution to each of these topics will be discussed. We present the results of a user case study and evaluate of the overall accuracy of our recognition process. Finally, we will discuss further development of CircuitBoard that could increase its completeness, scope, and usability in academia and industry.

RELATED WORK

CAD applications such as Logisim [5] and commercial Verilog/VHDL simulators such as ModelSim and VCS have been available for years and provide the sort of real-time debugging features we have outlined for CircuitBoard. These systems, however, are prohibitively unsuitable for draft work due to their formality and cumbersome, modal interfaces. These factors hinder the brainstorming process and are often the environments that drive engineers away from the computer and back to paper-based interfaces for their initial logic design work.

Digital logic circuit diagrams are well suited for automatic detection in sketch-based environments. There are a small number of well-defined symbols, with explicit relationships and behavior. While many researchers use logic circuits or other sorts of electrical circuits as examples of potential applications of their sketch-based frameworks [2,6,9] surprisingly few projects have been developed specifically geared toward this domain. SketchySPICE, an example CAD tool within the SATIN toolkit [7] recognizes logic gates but is intended to be a simple proof-of-concept application and is limited in its set of recognized gates. Alvarado et al. developed a more advanced system for sketch-based circuit recognition [1] that handles all types of logic gates. CircuitBoard is designed to incorporate more features than these applications, namely circuit simulation.

Liwicki and Knipping [8] developed a system that recognizes and simulates sketched logic circuits, provides a clock mechanism, and allows the user to consolidate and save a circuit as a labeled gate that can later be instantiated. Their system, however, constrains the user to a subset of logic circuits, specific stroke patterns for gate recognition (stroke must begin and end on the input side of the gate), and a modal input method. CircuitBoard aims to provide a modeless interface and a reduction of constraints on the user's input style is part of our core design philosophy.

Alvarado conducted two user-focused studies into the domain of sketched digital logic circuit design. [14] explores different methods of triggering gate recognition and providing recognition feedback, providing valuable conclusions into the expectations of users when they interact with a sketch-based system for logic design. [3] uses a large user case study to analyze real-world hand drawn logic gates and suggests several criteria for robust gate recognition based on the observed patterns and findings of the study. CircuitBoard uses strategies for gate recognition that heed Alvarado's conclusions and adheres to a user interface design principle that should be effectively targeted to its users.

SYSTEM OVERVIEW

CircuitBoard is written in Java 1.6 and uses the JPen library [15] for accessing the eraser features of the Tablet's stylus. Figure 1 is a screenshot of CircuitBoard demonstrating the simulation of a hand-drawn logic circuit. This screenshot was taken on our primary testing machine, an HP tx2000 Tablet PC running Windows Vista. The system can be broken down into four major components – the *user interface* (UI), which provides a minimal set of controls built upon research into practical sketch UI, *stroke segmentation* for the simplification of arbitrarily noisy input data, *gate recognition* for interpreting the drawn logic elements, and *circuit evaluation* for providing the debugging tools exclusive to our application.

USER INTERFACE

The primary philosophy behind CircuitBoard's user interface design decisions is providing a clean, minimal interface to the user. The simplicity of this interface provides the environment conducive to the type of work done on a whiteboard or sheet of paper. Alvarado et al.'s research into the expectations of users during their interactions with a sketch-based system [14] emphasizes that the system should stay out of the user's way until recognition functionality is requested. We will describe in this section how our interface adheres to this philosophy.

The first key design decision is the modeless process of composing logic circuits. The converse of this, for example, would choose one pen mode for gates, another for wires, etc. CircuitBoard does not require the user to specify a “mode” for types of input, rather, the high-level work of grouping sketched primitives into wires and gates is left to the system. This modelessness creates a consistent and unobstructed interface for draft work.

As CircuitBoard is intended to augment hand-drawn logic circuits with information, this visualization must provide data regarding the circuit's interpretation and simulation while adhering to the policy of not interfering with the user. We have concluded that there are two visual models of the

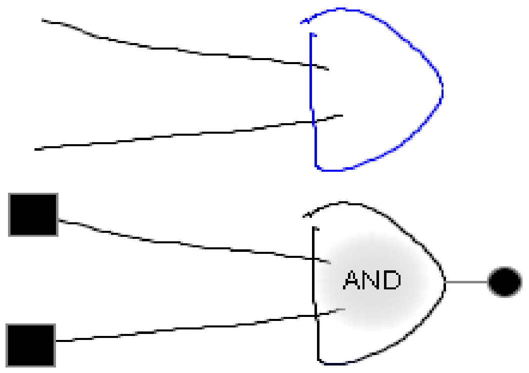

Figure 2. The top figure shows an AND gate being drawn and appended to two input wires. The coloring of the gate indicates that the users pen has not yet been released from the tablet surface. Once the pen has been released, as depicted in the lower figure, a label quickly fades in and out that states the type of logic gate that was interpreted. This automated feedback indicates to the user that recognition has occurred, and allows them to review the accuracy of the interpretation. Tapping the label before it fades out changes the gate to its next most likely interpretation.

logic circuit that are relevant to the user – reviewing the system's interpretation of the drawn logic circuit, and the debugging process itself. While CircuitBoard is modeless in its input, we have designed CircuitBoard to be modal in its visualization – the user can switch between "Normal" mode, "Gate" mode, and "Simulation" mode at any time to visualize their circuit's interpretation and simulation. The user can continue to edit the circuit as normal regardless of the current visualization mode.

"Normal mode" is the default mode of CircuitBoard. It's intended to provide a clean interface with minimal intrusiveness to encourage an unimpeded flow of thought. Each visualization mode is a superset of the functionality of "Normal mode" - the user is able to edit the circuit during any state of the system. Although the stroke segmentation system of CircuitBoard can handle the beautification of noisy stroke data, all sketched figures in CircuitBoard are drawn as-is, without beautification. Beautification has several negative impacts on the user's interaction with the system. The automatic nature of beautification removes control from the user, violating the philosophy of our UI design. Furthermore, the more "perfect" nature of beautified sketch fragments can make circuit elements seem 'finalized,' and can discourage the user from making changes or even questioning what has been drafted. [10] An exception to our aversion to automated, immediate feedback is shown in Figure 2. When a new logical gate is recognized, a label showing the type of recognized gate quickly fades in and out, to signify both the recognition of

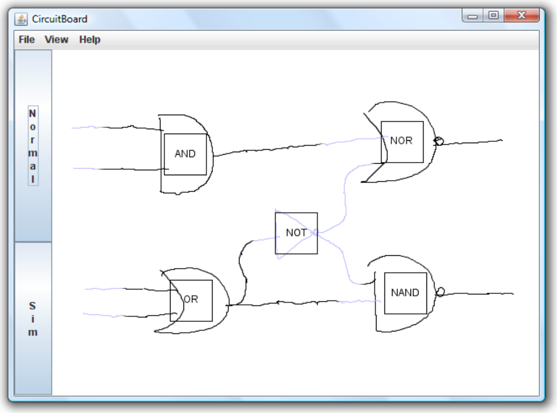

Figure 3. This screenshot of CircuitBoard shows the Gate visualization mode. The types of each logic gate are floated on top of their body within a bounding area that, when tapped, will change the gate to its next most likely interpretation. Signal flow through the circuit is being represented by a cascading color sweep from inputs to outputs that cannot be visualized well in a screenshot.

the gate and its type. Additionally, tapping the label when it is visible will change the value of the gate to its next most likely interpretation, based on CircuitBoard's gate recognition system. This provides a shortcut for users to quickly correct interpretation errors.

Figure 3 depicts "Gate mode", which primarily allows the user to review the correctness of CircuitBoard's interpretation of the hand drawn logic circuit. Each gate shown in the figure has a label on top of it displaying the type of the gate, as well as a box surrounding this label. Tapping this box will also change the gate to its next most likely interpretation, based on the gate recognition system. The second feature of "Gate mode" allows the user to visualize the signal flow through the logic circuit. Cascading colors travel through the wires of the circuit from inputs to outputs, a still frame of which can be observed in Figure 3. These two forms of visualization allows the user to review and correct the interpretation of the logic gates, as well as review the connectivity of the circuit and observe unintended wire omissions or system misinterpretations.

"Simulation mode", as shown in Figure 1, is the primary feature of CircuitBoard. The square figures on the left side of the figure attach themselves to the free inputs of the circuit, and the circular figures on the right side on the figure attach themselves to the free outputs of the circuit. Elements of the circuit drawn in black evaluate to 0 or false, and elements drawn in red evaluate to 1 or true. Interacting with the circuit involves toggling the circuit's inputs, which is achieved by simply tapping the input areas with the stylus. While other systems that have implemented automated circuit evaluation requires the user to draw a 1 or

0 next to circuit inputs, [8] our system provides a natural, and intuitive manner of interacting with the circuit that is conducive to informal review and debugging of the logic circuit design.

Several methods of changing visualization modes are made available to the user. The most notable are the large buttons to the left of the drawing pane. These buttons change their functionality based on the current visualization mode, such that the two buttons always provide transitions into the two alternate modes. While research into the practicality of WIMP-based (Windows, Icons, Menus, Pointers) elements in sketch-based systems, such as buttons, suggest alternatives to their use, [4] we feel that only the size and subsequent accessibility of traditional WIMP interactions make them unsuitable for sketch-based applications. Large buttons as used by CircuitBoard can still be accessible, even with a stylus. While gestures for changing modes are possible alternatives to buttons, gestures in the context of CircuitBoard could not be robustly disambiguated from potentially intended circuit elements. Furthermore, the physical presence of the buttons on the screen serve to both remind the user of the available tools to assist their draft work and free the user from needing to remember program features and arbitrary pen gestures.

In addition to the on-screen buttons, the user has the option of using keyboard commands to change modes. As the visualization modes are understood as temporary visual augmentations to the drafting of the circuit, modifier keys such as Control and Shift match this convention. Holding down Control will allow the user to briefly review gate mode for quick examination, and holding down Shift will allow the user to briefly view the simulation of their logic circuit. Releasing these keys allows the user to return to an unobstructed design environment. As debugging a circuit can require more extended interaction, Caps Lock will toggle Simulation mode, to complement the understood functionality of the Shift key.

RECOGNITION PROCESS

The critical element of CircuitBoard is the system that recognizes the sketched digital logic gates and interconnecting wires that comprise the logic circuit. Our solution involves two steps: a scale-space based algorithm introduced by Sezgin et al. [12] to segment the inputted pen stroke data into simplified fragments, and a computer vision algorithm using shape contexts to match groups of stroke fragments to digital logic gates.

CircuitBoard is able to recognize the full set of logic gates – AND, OR, NOT, NAND, NOR, XOR, and XNOR. These gates can be drawn with any dimensions, and in any number of strokes. There are no temporal, directional, or stroke order constraints on the drawing of gates, providing the user the maximum amount of flexibility. The recognition process executes when an input stroke is

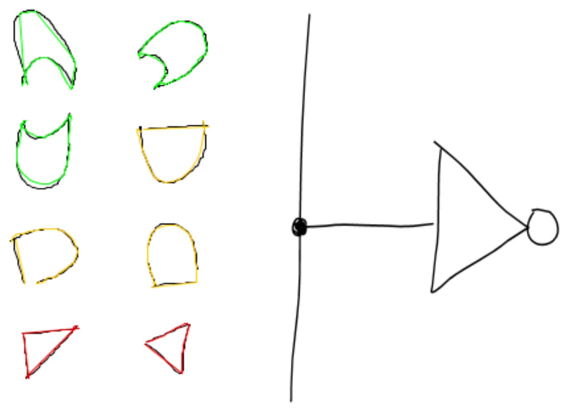

Figure 4. Complex cases of digital logic design to be handled in the near future by CircuitBoard. The left figure shows examples of arbitrarily rotated logic gates. CircuitBoard is able to accurately identify these gates, and the coloration of each of the eight figures indicate the correct interpretation. Robustly detecting their orientation, however, requires further development. The right figure shows how wire splitters introduce ambiguity to an otherwise well defined domain – in this example, the top and bottom endpoints of the wire could equally be considered the "input", where a control should be provided to the user in "Simulation" mode for debugging purposes.

complete; when the pen is released from the tablet surface. The responsiveness of the system is largely dependent on the length of the input stroke as opposed to the complexity of the scene. However, lengths that would cause noticeable latency (complex polygons, extended polylines, etc.) are not common in the domain of digital logic circuits. In the average case, with complex scenes with many logic gates, wires, and orphaned strokes on the screen, the common latency time is ½ to 1 second. Although this number can seem high, the latency does not interfere with the pen down event that begins the next stroke. This allows the user to chain one stroke into another without the system imposing a need to pause. Additionally, this latency time usually elapses before the next step in the logic circuit drafting process has been decided or begun by the user. CircuitBoard also handles cyclic digital circuits, and is able to detect unstable logic configurations. Erasure is supported that makes use of the stylus' eraser, providing a familiar method of correcting input or design mistakes.

CircuitBoard has a small amount of basic features that remain incomplete. We are able to accurately test for the presence of arbitrarily rotated logic gates, but detecting the angle of their orientation requires further development time. The left half of Figure 4 shows examples of our recognition system on eight correctly identified gates in arbitrary orientations. In the meantime, CircuitBoard assumes gates are pointing to the right.

Additionally, the splitting of wires to provide inputs to multiple gates requires a complex approach to reach a satisfiable level of robustness. While logic circuits are well defined in their operation, splitting wires introduces ambiguity to the system – certain areas of circuits, or even entire networks of wires, can have no well defined input. The right half of Figure 4 shows one such case, where two possible inputs to the circuit are present, where in "Simulation" mode only one controllable input will be expected. While work has been made in implementing this feature, further integration, debugging, and handling of many potential degenerate cases will need to be done before this can be complete.

STROKE SEGMENTATION

The first step in gate recognition is to simplify and segment stroke data inputted into the system. Pointer-based input is inherently noisy, due to machine imprecision or the imperfect muscle control of the user. Although the user may intend to draw a straight line, without a filter to interpret the stroke data on a high level, the system will only be able to interpret this raw stroke data as an erratic set of points, with potentially large amounts of local curvature at points along strokes intended to be straight. Sezgin et al.'s scale-space based algorithm [12] filters out this noise by using a Gaussian filter and segments strokes into line segments and 3rd order Bézier curves.

Sezgin et al.'s algorithm graphs the curvature of the stroke against the sample points of the stroke, and applies a large range of Gaussian filters with increasing σ values. For each filtered curvature signal, local maxima are chosen to be the feature points of the signal and a graph is made of the amount of feature points against the range of σ values. A final σ value is chosen by splitting this feature point count vs. filter range graph into two ranges, fitting an ODR line to each region, combining the orthogonal least squares error for both lines, and finally choosing a σ value that minimizes this error. The feature points of this filtered signal are used to segment the stroke, and each segment is individually classified as linear or curved.

Classifying the segment under Sezgin et al.'s algorithm considers the Euclidean distance between the endpoints of the segment against the total length of the stroke between these points. For linear segments this ratio will be close to 1, and along curved segments this ratio will be noticeably larger than 1. Sezgin et al. describes in an earlier paper [13] a method of fitting a 3rd order Bézier curve to curved segments. The curve can also be recursively decomposed into smaller, tighter fitting Bézier curves to minimize error to a desired threshold.

A major issue in implementing Sezgin's scale-space algorithm involves the amount of pen accuracy generated by the system. Figure 5 in [12] shows Sezgin's feature point count vs. Gaussian filter strength graph generated by a simple drawn figure. At lower filter strengths, the order of 100 feature points are detected in the signal. This granularity results in a well formed curve for the line fitting process that determines the final filter strength. However, in our implementation, the average feature point count of a logic gate with minimal filter strength is on the order of 15. This number does not lend itself well for a rounded curve such as Figure 5 in [12], and ends up being inconducive for accurate line fitting. The result is that stroke segmentation is often unreliable.

The primary source of this problem comes from tablet polling rate. Test runs would pass the first dozen input strokes as properly segmented, but would begin failing without exception as the test would continue. Our conclusion was that the amount of time taken to draw the scene using standard Swing components was beginning to impede on the tablet's ability to poll the position of the stylus. Our solution was to batch the visual representations of all gates, wires and orphaned line fragments into an offscreen buffer that could be drawn once to the screen instead of calling the drawLine() function $k*n$ times for n gate, wire and orphaned line segments, with k as the amount data points per element. Implementing this system in a hardware accelerated framework such as OpenGL would be an even better solution, but this was not possible due to time constraints on our project.

Various tweaks to Sezgin's algorithm increased its accuracy in the domain of sketched circuits (which ideally consist of very few fragments). We check the height of local maxima in the filtered curvature graph as a percentage of the global maximum of the signal, and do not accept a feature point if this percentage is below a certain threshold. We also discard feature points near both ends of strokes, where tiny accidental pen flicks upon lifting or setting down the pen would often occur.

Additionally, choosing the first, k^{th}, and last points on the graph to find two lines on the filter strength vs. feature point graph, as opposed to fitting two ODR lines within these regions, resulted in more accurate results.

Once CircuitBoard segments the input stroke, and classifies each segment as linear or curved, adjacent colinear lines can be combined if their slopes are within a certain threshold. Finally, this list of fragments is then to the gate recognition system.

GATE RECOGNITION

At this point of recognition, the scene consists of a list of input strokes decomposed into linear and curved segments. The temporality of these events is preserved by listing them in the order in which they are received. Our gate recognition system will analyze this list of events and determine the presence of logic gates.

Search for gates

As the recognition process and subsequent gate identification will be triggered by the most recent stroke event, we can assume that at least one of the fragments of the recent stroke belongs to the gate that will be recognized. As the user should be allowed draw gates and wire segments within a single stroke, we cannot assume that the beginning or final segments of the stroke will belong to a gate. The first step of this process is to detect a closed shape. We consider each fragment of the recent stroke event, and scan backwards through the list of events and fragments to locate a closed form. We also consider the 'closeness' of the ends of temporally adjacent fragments based on the size of the fragments, as opposed to a fixed threshold. This allows closed shapes to be detected at any scale.

Once a closed shape has been found we send it to the classifier, which matches the shape to training data for AND, OR and NOT gates. This returns the errors of the figure for each type of gate. If the error of the best matching gate is within a certain threshold, the appropriate gate is instantiated and the consumed fragments removed from the list of events. This allows other fragments of the recent stroke event to form other circuit elements. Since other gates types are composed of AND and OR gates, we do not test for them at this juncture.

To recognize XOR gates we test for a series of fragments parallel to and within a certain range of an OR gates input side. The new XOR then consumes these elements and replaces the OR gate in the scene. Recognizing negated gates (NAND, NOR and XNOR) involves detecting bubbles within a bounding box attached to the output end of an invertible gate which is scaled relative to gate's size. CircuitBoard accepts NOT gates as triangles without a bubble on the output, allowing users to casually represent them in this manner. We note that an orphaned NOT gate represented as a triangle can be intended to be oriented in three possible directions, and a bubble attached to a NOT gate can resolve this ambiguity.

Classification

As there are many ways in which users can draw a single logic gate, and because of the flexibility our system provides for how the user can draw their circuitry, using the types of fragments (line segments and curves) that make up the gate's structure is insufficient to classify it. CircuitBoard would need to be custom fit to many different styles of drawing gates, and could never handle the full range of styles. This would also hinder our ability to handle user error. We instead use a computer vision based method on the closed shape, focusing only on the shape of the figure independent of the way it was drawn.

We use a modified version of the Shape Context method, described in [11]. Mori et al. introduce the idea of using shape context in object recognition, to quickly search a series of figures for similar shapes. The shape context

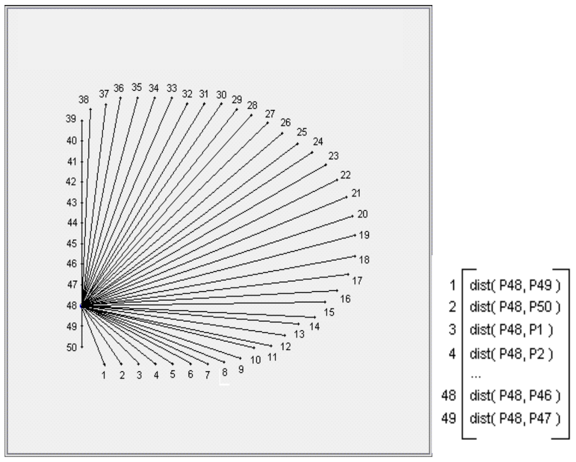

Figure 5. The shape context of an AND gate at point number 48 out of 50. The shape context algorithm considers the distance from each point to every other point on the shape. The values of the shape context vector at each point consist of these distances starting from the point "to the right". In this figure, the distance to point 49 would be the first value of the vector, followed by point 48's distance from point 50, followed by the distance from point 1, and so on. The shape context vector is shown to the right of the image.

algorithm considers n discrete points on the shapes outer contours, as well as any inner contours if present, sampled at equal intervals to evenly space the samples. For each of these points, the distance to every other n-1 points is stored in a histogram for that point. When a shape is compared to a set of precomputed shapes (in our case, pretrained AND, OR and NOT gates) a subset of points on the query image are chosen; the histograms of these points are compared to those of the precomputed shapes, and a certain error from the original shape is determined. Outlying interpretations with high error can be culled against a given threshold, and other algorithms can be used for the remaining set of matches that could better differentiate between closely matched interpretations.

In CircuitBoard we use a vector of distances instead of a histogram at each point. This vector stores the distances from the given point to all other points of the shape. This modification of the algorithm provides invariance to rotation. An example of this for an AND gate is shown in Figure 5. We note that this is only possible due to the logic gates consisting of shapes of a single contour. To make our description invariant to scale, we normalize this shape context vector. We also note and accept the need for $O(n^2)$ space required for the context vector for each shape.

To compare a candidate to the training samples, we select k points on the candidate figure that are evenly spaced along its ordered points. For every training sample, we find the best fitting points to each of these k points on the candidate figure, and generate an error by taking the average difference between each pair of vectors found by a summation of the difference between each element. We also reverse the shape contexts of the training data to handle gates outlined by the gate searching stage in the opposite direction. This step involves a total of $2 \times k \times n$ comparisons.

We then sort the training samples in ascending order with respect to their error with the candidate figure. The first sample in this list will be the most likely gate interpretation, with the remaining elements of the list used by the other systems of CircuitBoard to determine the "next most likely" interpretation.

CircuitBoard uses $n = 50$ and $k = 10$, which results in satisfactory results with no noticeable latency. Increasing n and k could result in better gate recognition at the cost of speed, and is something yet to be experimented with. We also provide a way for the user to train the system to their own handwriting if the default training samples do not function well for that user. Currently, 9 training samples are used, 3 for each basic gate type (AND, OR and NOT).

Handling rotation

Our shape context implementation provides a way to match two figures invariant to rotation, translation, scale and distortion. However, after two figures have been matched (again, only be choosing the figure of a minimal error) this shape context method gives us no information about the angle at which the gate is orientated. Although the fragment types that make up the gate (line segments and curves) could be used to determine the orientation of the gate, this would require a second recognition process, facing the same problems as described in the previous section.

Until this problem is solved, our system assumes that gates are facing to the right once they have been detected. Our newer approach which remains to be tested will select k points on the training sample not by best fit, but also spaced evenly along the figure. Additionally, two points on the training figure can be marked that denote the corners of the input side of the gate. The selection of k points on the candidate figure can then be rotated around the figure, and the minimal error for the given rotation will be chosen as the best fitting match. The points denoting the corners of the candidate gate's input side can then be identified.

CIRCUIT EVALUATION

Gates are then sent to the circuit evaluation system, which maintains a graph of the circuit. Gates and wires alike are treated as nodes of the graph, simplifying the design of the system by having each node manage its own input and output constraints. This design applies well to the logic circuit domain, where, for example, NOT gates and forked wires accept only one input and can have many outputs, and wires can only take one input and one output. As each node manages the status of their inputs and outputs, the inputs and outputs of the overall circuit utilized in "Simulation" mode can be quickly found with a linear scan through the circuit's gates.

After every stroke event, depending on whether a gate or wire was created, the list of orphaned stroke events is checked to see if any strokes are positioned to be promoted to a wire. CircuitBoard also handles the event in which this process can recursively generate a large network of wires.

The logic circuit being drafted is evaluated using an iterative process with a dirty/clean flag on each node of the graph. An update method called on each node checks that node's inputs and updates the output value based on the node's type (AND, OR, XOR, etc.). This update method will also set the node's flag to clean. If a node's output is changed by the update process, each of its output nodes has its flag set to dirty. One iteration of the circuit evaluation process will identify all the dirty gates of the circuit and update them. Each subsequent iteration will identify and update until no dirty gates are identified in the circuit, or until some maximum amount of iterations is reached which indicates an unstable circuit.

The worst case amount of iterations for a stable circuit to converge without cycles is n, where n is the amount of nodes in the graph. This is the case given a linear circuit where in each iteration, the i^{th} node, starting at the input end of the circuit, is the final node to be evaluated and sets its outputs to dirty. This causes the value to propagate down the circuit, one node at a time, over the course of n iterations. Evaluating the bounds of a cyclic circuit is beyond the purposes of this paper, but is predicted to be at least $O(n^2)$, as some circuit could possibly be devised where one such "sweep" through the graph would set one node at a time to be stable.

USER EVALUATIONS

Our evaluations of CircuitBoard presently consist of an open demonstration and discussion of the system with available forms for impressions and suggestions, and a more formal case study involving PhD students from ArchLab, a computer architecture lab at UCSB.

Our open demonstration asked our reviewers for their thoughts on the intuitiveness of our user interface, the usefulness of the visualization modes, the accuracy of our system as well as the usefulness of CircuitBoard as a whole. The verbal responses at the demonstration were favorable, and six observers who were also involved in sketch-based development participated in our review.

Users found the system's interface to be intuitive, remarking on the recognition notification, utility of viewing the signal flow, simplicity of interacting with the simulation, and modeless drafting interface. Their review of the systems accuracy was positive, yet indicated a need for further work – certain elements of wiring up the circuit was described as "finicky", and the types of the drawn gate were sometimes misinterpreted. Handling gates of different rotations was also requested.

The usefulness of the program was quite positive. While our reviewers were limited in their history in logic circuit design to the classroom, they noted the advantage CircuitBoard could provide to themselves and other students in digital logic courses. Our users also expressed a desire to be able to export their design to some sort of code or a hardware description language (HDL), and preferred our system to a traditional CAD application for circuit design.

The case study at the ArchLab asked users to attempt to sketch a set of gates before and after training, to determine the effectiveness of personalized training data. We also asked users to input a series of five increasingly complex pre-defined circuits into CircuitBoard, to determine the effectiveness of our system in practice. Two members of the lab were available, given time constraints, to participate in our study.

One observation of the study found that the accuracy of gate recognition improved as users began to grow accustomed to the system. Personalized training data made a significant difference in the system's ability to recognize the user's input. While the system was shown to need much improvement in its accuracy for users unfamiliar with CircuitBoard, some of the final tasks the users attempted saw much higher accuracy than initial tasks. Many elements of user input styles were observed that could directly improve our algorithms and approach to the problem, such as the order in which XOR gates were drawn, and the shape of OR gates. We also noted that the users preferred to draw within one visualization mode (the first user in gate mode, the second in simulation mode) as opposed to using these modes for temporary review and verification of the design.

Users did find drafting circuit gates on a tablet PC to be and obstacle that we did not previously consider. One noted that they did not want to damage the tablet screen by using the stylus like they would a pen or whiteboard marker, which could affect their interactivity with the system. One user noted that the lowered friction between the stylus and the screen, when compared to that of a pen and paper or felt-tip marker and whiteboard, made it harder to stop the stylus and end strokes at desired points. The users did, however, find the eraser functionality of the tablet stylus to be intuitive and useful, and acknowledged the utility of the system, despite the unfamiliar interface.

As our preliminary user evaluations have been limited, more extensive user evaluations could be conducted in the future. Further development of the system to enhance the robustness of CircuitBoard's features will be needed, as they greatly affect user impressions of the system.

CONCLUSION AND FUTURE WORK

We have discussed the benefits of paper- and whiteboard-based interfaces for digital logic circuit design, and noted the benefits of CAD environments for circuit simulation and debugging. We proposed a sketch-based CAD environment that combines a natural way of expressing complex logic circuits with a simulation environment conducive to debugging such a design. While our implementation has several features that remain to be completed or improved in their robustness, CircuitBoard breaks new ground in digital logic circuit CAD applications in terms of flexibility and usability.

Once the current feature set of CircuitBoard is complete, there are more components that could be valuable additions to CircuitBoard's functionality. AI based algorithms can be introduced to expand on detecting the users intentions to draw circuit elements, which should assist the accuracy of the system. Erasure on a per-pixel level should be supported, and we would like to experiment with a form of erasure that leaves behind faint figments of the previous stroke, not unlike how a pencil eraser will not completely remove a pencil stroke. A system such as this can provide a temporal progression of the design as opposed to a complete elimination of previous design decisions from sight and mind. We would also like to see CircuitBoard simulate the behavior of logical gates, in terms of propagation delay, rise time, fall time etc., to support the playback of intentional or unintentional behavior reliant on these additional factors. The addition of a configurable clock once this modification is implemented would be a natural evolution to the system. CircuitBoard could also be expanded to make use of a multi-touch, whiteboard sized display, to enable multi-user collaboration on logic circuit design.

Finally, features that would be critical to the scaling of CircuitBoard to systems with complexity outside that of the classroom would be the ability to create custom composite circuits that could be labeled and instantiated by simply drawing a box around a logic circuit and writing the circuit's label within it. This object oriented design, which could allow dynamic creating and editing of circuitry could be done in a bottom up or top down manner – a box can be drawn labeled “Processor”; an editing mode for this custom circuit could be entered and boxes labeled “Control”, “Data Path”, and so on could be placed within it, etc. Finally, designs created in CircuitBoard could be exported to Verilog or VHDL for use in any system that handles digital logic circuitry.

ACKNOWLEDGMENTS

We'd like to thank Professor Timothy Sherwood and Ryan Dixon for the instruction, guidance and inspiration involved in the development of CircuitBoard.

REFERENCES

1.Alvarado, C. Sketch recognition for digital circuit design in the classroom. *Proc.* Invited Workshop on Pen-Centric Computing Research *2007.*

2.Alvarado, C. and Davis, R. SketchREAD: a multi-domain sketch recognition engine. Proc. UIST 2004, 23-32.

3.Alvarado, C. and Lazzareschi, M. Properties of Real-World Digital Logic Diagrams. *Proc. PLT 2007,* 12.

4.Apitz, G. and Guimbretière, F. CrossY: a crossing-based drawing application. *Proc. UIST 2004,* 930-930.

5.Burch, C. Logisim: a graphical system for logic circuit design and simulation. *Journal on Educational Resources in Computing* 2, 1 (2002), 5-16.

6.Gross, M. and Do, E. Ambiguous intentions: a paper-like interface for creative design. *Proc. UIST 1996,* 183-192.

7.Hong, J. and Landay, J. SATIN: A toolkit for informal in-based applications. *UIST 2000,* 63–72.

8.Liwicki, M. and Knipping, L. Recognizing and Simulating Sketched Logic Circuits. *Proc. KES 2005,* 588-594.

9.Paulson, B. and Hammond, T. PaleoSketch: accurate primitive sketch recognition and beautification. *Proc. IUI* 2008, 1-10.

10.Plimmer, B. and Apperley, M. INTERACTING with sketched interface designs: an evaluation study. *Proc. CHI 2004*, 1337-1340.

11.Mori, G., Belongie, S. and Malik, J. 2001. Shape Contexts Enable Efficient Retrieval of Similar Shapes. *Proc. CVPR 2001,* 723.

12.Sezgin, T., and Davis, R. Scale-space based feature point detection for digital ink. Proc. Making Pen-Based Interaction Intelligent and Natural 2004.

13.Sezgin, T., Stahovich, T. and Davis., R. Sketch based interfaces: Early processing for sketch understanding. *Proc. PUI 2001, 1-8.*

14.Wais, P., Wolin, A. and Alvarado, C. Designing a sketch recognition front-end: user perception of interface elements. *Proc. SBIM 2007,* 99-106.

15.Wendt, M. et al. JPen – Java Pen Tablet Access Library. [Online]. Available: http://jpen.wiki.sourceforge.net/