Properties of Real-World Digital Logic Diagrams

Michael Lazzareschi
Senior Project in Computer Science
Pomona College

Advisors:
Christine Alvarado
Tzu-Yi Chen

May 1, 2006

Contents

1 Introduction

Originally, technical diagrams in private industry and education were drawn only on paper. While this method of drawing is convenient for creating diagrams, it makes analyzing the diagrams challenging. With the advent of computers came the prospect of automating many tasks relating to diagrams that are tedious to perform by hand. Once diagrams are stored in computers, it is possible to add information about the names of each symbol in the diagrams. If the individual symbols in diagrams are identifiable to the computer, the diagrams can be analyzed efficiently. For example, if an engineer draws an electrical circuit diagram and the computer is able to recognize each symbol in the diagram, the engineer could run a simulation program on the diagram to test if the circuit performs the intended function. Without the use of computers, this task would have to be performed visually, requiring more time, and, introducing the possibility of human error. Initially, computers provided this advantage in diagram analysis at the cost of the ease of drawing that a pencil and paper provide. The simplest methods of diagram entry required extensive use of the computer mouse. However, during the time that computers have been used to store diagrams, the methods of inputting diagrams have evolved. The invention of digitizing tablets and pens provided a great advancement in the ability of engineers and academics to input diagrams to computers. Using a digitizing tablet and pen, users can draw naturally and have their diagrams stored on computers without any additional difficulty.

Ideally, the process of labeling each symbol in a diagram, known as diagram recognition, would be fully automated so that the user need not specify the identity of a symbol every time he or she draws it. An effective diagram recognition system that allows the user to draw naturally, without requiring any additional labeling input from the user, must rely on information about the ways that people draw diagrams to make decisions about how to label each symbol. In this paper, we analyze digital logic diagrams drawn by students in a real hardware engineering course in order to gather information about how the symbols are drawn. To better understand the need for diagram-specific information, we must first introduce the process and challenges of diagram recognition as well as the basics of the digital logic domain.

1.1 Digital Logic Diagram Recognition

Diagram recognition is the process of taking a diagram and labeling each meaningful symbol with the correct name. This process has been approached using different methods, but

all methods include two basic steps: segmentation and symbol identification. During segmentation, groups of *primitive components*, often line segments and arcs, are identified as potentially meaningful symbols. In symbol identification, potential symbols are labeled with their names. The result of the whole process is a labeled version of the diagram. While the set of potential symbols resulting from segmentation may contain groups of primitive components that overlap, the final result after the symbol identification step must be a consistent mapping of primitive components to labeled groups. Segmentation and symbol identification may be approached discretely and successively as in [4] or simultaneously as in [1]. For a more complete discussion of recognition systems see [1, 4, 6].

To perform these two steps, recognition systems take advantage of a variety of assumptions about how diagrams are drawn. For instance, during segmentation, it may be useful to make an assumption about the range in numbers of primitive components that compose symbols. If we know that no symbol contains more than five primitive components, we can decrease the number of potential symbol groups by removing groups containing more than five components from consideration. During symbol identification, it may be useful to make assumptions about both the number and type of primitive components that compose each type of symbol. For instance if we are trying to recognize triangles, we know that they are comprised of three line segments, so we can determine that any group of components that does not consist of three line segments cannot be a triangle symbol.

In designing a recognition system, it is essential to identify assumptions that, when made, will provide accuracy in recognition without imposing *unnatural* restrictions on the way in which users draw. Assumptions that place unnatural restrictions on drawing include any assumption that forces the user to avert his or her attention from drawing in same way he or she would on paper. For example, consider a recognition system that requires users to pause after drawing a symbol, and then click a box specifying the proper label for the symbol. While this system may perform recognition with perfect accuracy, it forces the user to draw in an unnatural manner. On the other hand, assuming that every symbol was drawn using at least one stroke of the pen does not impose any restrictions on drawing. However, because this assumption is so basic, it does not aid recognition.

Good recognition systems adopt assumptions that improve recognition accuracy by imposing *natural* restrictions on users. When drawing family tree diagrams, such an assumption might be that users draw marriage connections around the same time as they draw the symbols for the people who are married. While some users might draw all the marriage connections once they have drawn all the symbols for people, the assumption holds true for

the vast majority of users because they naturally draw this way; for most users, because of this assumption, the system can gain accuracy without imposing unnatural restrictions.

One assumption common to current recognition systems is that the type of diagram on which a given recognition system operates is restricted; a diagram recognition system is designed to operate in a single diagram domain because the problem of universal recognition (across all domains) is too hard. In this study, we focus on gathering information within the domain of digital logic diagrams. Figure 1 shows the set of standard digital logic gates that appear in diagrams. These gates are connected by line segments that represent wires. As can be seen in Figure 2, a sample digital logic diagram, typical diagrams consist of one or more gates connected by wires. There is at least one input wire and one output wire to the whole diagram. These inputs and outputs are often labeled with text; in Figure 2 the inputs are labeled A, B, and $CarryIn$ and the output is labeled $CarryOut$.

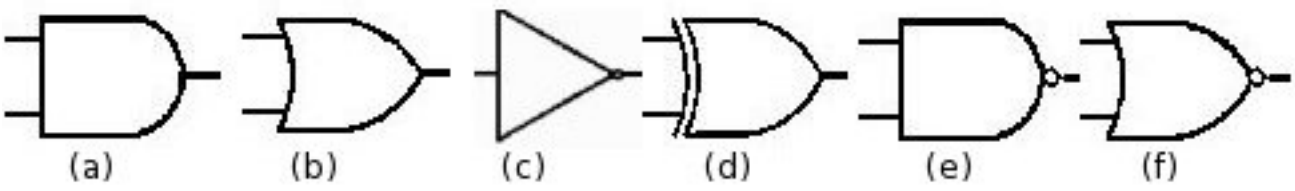

Figure 1: Digital Logic Gates: (a) AND (b) OR (c) NOT (d) XOR (e) NAND (f) NOR

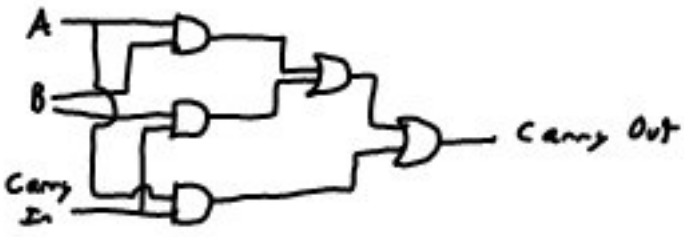

Figure 2: A Digital Logic Diagram

We study how digital logic diagrams are drawn in order to identify good assumptions. This is a difficult task due to the variety in drawing techniques that are possible. While it is possible that a user could draw diagrams in an ordered, logical manner, with one stroke for each primitive component, it is also possible that a user could create diagrams with identical appearances by drawing a series of points in random order, or in any order using

any number of strokes. The greater the variety in users' drawing methods, the harder the task of recognition because the underlying complexity of the process that a diagram recognition system utilizes to make labeling decisions reflects the variety of ways in which the symbols are drawn.

The domain of digital logic diagrams presents unique challenges to recognition because of the nature of the symbols. Many of the gates are similar in appearance. For instance the differences between the form of an AND gate (Figure 1(a)) and the form of an OR gate (Figure 1(b)) are only subtle variations in curvature. Even more striking is the difference between the form of an AND gate and the form of a NAND gate (Figure 1(e)). The inclusion of a small circle changes the function of the gate entirely. The difficulty of recognizing symbols that have only subtle differences in form is compounded by the fact that people often draw sloppily. For instance in Figure 2, the right sides of the OR gates are curved, whereas the proper form of an OR gate would be more pointed as in Figure 1(b).

In addition, the fact that the form of wires is not defined by a consistent shape leads to a great variety in possible drawing styles. For example in Figure 2, some wires such as the wire representing the input A, are straight lines, while other wires are drawn with right-angled jogs to allow them to connect two gates that are not aligned horizontally. Other wires include curved sections to denote the fact that they are meant to pass over the wires beneath the curved sections rather than intersect with them. While including curved sections in wires to indicate a lack of intersection is the correct method, it is common for users to omit curved sections because humans can answer the question of whether two wires that cross in a diagram are meant to intersect in the circuit by looking at the surrounding context. Similarly, it is not unusual for students who are less practiced in drawing digital logic diagrams than engineers, to draw curved wires in order to connect two gates that are not aligned horizontally rather than including right-angled jogs as in Figure 2. Finally, wires introduce complexity to the task of recognition because the symbol for a single wire in a circuit might be drawn using multiple strokes, which were drawn at very different times. Often, in large diagrams, users will draw part of wire when they finish drawing a gate to denote the output of that gate, draw some other part of the diagram, and then come back to the half-drawn wire once they are ready to connect it to the gate to which it is an input. In these cases, a recognition system must rely on other information than temporal proximity of strokes to determine which strokes form a single wire. Because so much variety exists in diagrams, we must examine them to determine how they are commonly drawn, and, using this information, determine assumptions that allow the user to draw naturally.

1.2 Gathering Diagram Specific Information

In this paper, we begin the process of designing a digital logic diagram recognition system by gathering information about how the diagrams are drawn. To this end, we address five questions:

1. Are gates drawn with consecutive strokes?
2. What is the average number of strokes that compose each type of gate? Specifically, are gates generally comprised of more than one stroke?
3. Are wires generally comprised of one stroke?
4. When do users draw gates relative to the input and output wires that connect to them?
5. Do bounding boxes fit tightly around groups of strokes composing gates have higher percentages of inked pixels than similar bounding boxes fit to groups of strokes including wires?

We chose these five questions after considering several others because each is a question that can feasibly be answered, and whose answer will assist during recognition. The answers to questions 1 and 4 are important during segmentation because, if there are trends in the order in which users draw symbols, it may be possible to take advantage of them when forming potential symbol groups. Similarly, the answers to questions 2 and 3 may be useful in devising a segmentation scheme because they will provide a maximum and minimum number of strokes that compose potential symbol groups. Questions 2 and 3 are also important to the symbol identification step because it will be useful to compare how many strokes are in a potential symbol group and the average number of strokes in each type of symbol. The answer to question 5 may be helpful to segmentation as well because, if groups of strokes that have higher *ink density*, meaning that they compose gates that have a higher percentage of inked pixels in tight bounding boxes than other groups, the segmentation scheme can give preference to symbol groups which have high ink density when creating potential symbol groups.

2 Methodology

2.1 Study Details

We pursue the five questions by analyzing data that came from real-world digital logic diagrams. Our data consists of thirty-two diagrams drawn by ten students in Engineering 85 at Harvey Mudd College, a digital electronics and engineering course (for more information about this course see [5]). Each subject was given a tablet computer that he or she agreed to use in class to take notes, and outside of class, to work on course assignments. The main goal in data collection is to ensure that the data is authentic in order to glean information that is useful for designing a recognition system with high performance on diagrams drawn without instruction. The subjects were asked to use the Windows Journal© program to take notes because this program has many features that make taking notes feel natural. While Windows Journal© does have recognition features, no recognition was performed while the subjects took notes. In addition, because we want data that was created naturally, in the course of work, we did not give the subjects any special instructions about how or when they should draw diagrams. For this reason, different subjects drew different numbers of diagrams in the week during which we collected data.

2.2 File Conversion

The original notes taken by the subjects were contained in jnt (Windows Journal© format) files. In order to answer our research questions, we developed semi-automated techniques to analyze the data. These techniques need access to information about which points of ink compose each stroke and information about when each stroke was drawn. In addition, the questions concerning the order of strokes in gates and the number of strokes in symbols (questions 1, 2, and 3) rely on information about the labels of groups of strokes that form symbols. For example, to determine the average number of strokes in gates, each gate must be labeled with its type. While the other questions (4 and 5) also concern labeled groups of strokes, they require that the diagrams are labeled differently than the first three questions. Question 4 concerns the order in which input and output wires are drawn relative to the gates to which they connect. To answer this question, not only does each symbol need to be labeled, but also the symbols to which each symbol connects need to be labeled as adjacent symbols. To answer question 5, the diagrams need to be labeled even more extensively; groups of strokes that do not even compose symbols must be labeled as symbols in order

to compare their properties with groups that do form symbols. Because information about the strokes in diagrams is not easily accessible in jnt files and it would be difficult to label symbols in jnt files, we converted the diagrams to more convenient formats.

We extract the diagrams from Windows Journal© notes, perform a series of procedures to convert the diagrams into formats that are conducive to analysis, and hand label the symbols in each diagram. Because file format conversions are intricate and require precise knowledge of the formats involved, we adapt existing systems for conversion. Table 1 shows the file format conversions that are used on the diagrams, in the order that they are performed, as well as the program used to perform each conversion, which questions are examined as a result of the conversion, and the original author of the program used for conversion.

Table 1: Different Diagram File Format Conversions in Order of Use

From Format	To Format	Program Used	Question Numbers	Original Author
jnt	Journal xml	Tablet Journal		C. Chesnut
Journal xml	ink	Table Journal	4, 5	C. Chesnut
ink	ink.gif	Data Collector		MIT CSAIL
ink.gif	drs	InkConverter		MIT CSAIL
drs	MIT xml	drsXmlConverter		MIT CSAIL
MIT xml	labeled xml	labeler	1, 2, 3	MIT CSAIL

2.3 Accessing Information About the Order and Labels of Strokes

To examine the questions about the order of strokes in gates and the number of strokes in symbols (questions 1, 2, and 3), we must gather basic information about the strokes in diagrams as well as information about the labels of strokes. The first step is to convert the diagrams to labeled xml. Initially, the diagrams are located in Windows Journal© documents, which may also contain hand-written notes, typed text, and images. The information in these files is in a proprietary format, but fortunately Microsoft® provides functions that convert jnt files to Journal© xml files, in which the information is more accessible. This step is performed using Tablet Journal, an open-source utility provided at [3]. Once a Journal© xml file has been created, Tablet Journal is used to *deserialize* the desired page in the file that contains a diagram. In the deserialization process, the information about what is contained on that page is extracted from the xml and stored in C# objects. We altered Tablet

Journal so that as the xml is being deserialized, it creates a C# *ink* object that stores information about each stroke of the pen including the time that it was drawn and which points it includes. Using this feature, each page in Journal© xml files that contains a digital logic diagram is converted to an ink file.

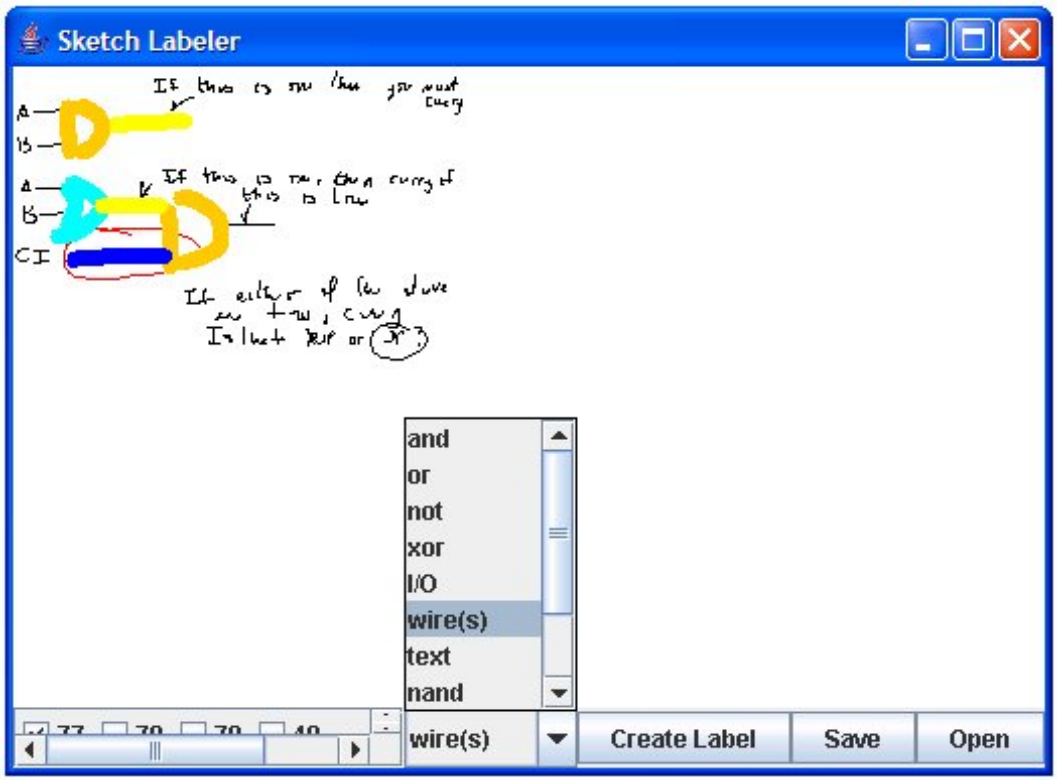

Figure 3: A Partially Labeled Diagram

Once the pages containing diagrams are stored in ink files, the remaining steps in converting the diagrams to labeled xml are performed using utilities created by M. Oltmans and A. Adler from [7]. First, the ink files, each of which contains information about all the strokes from a page of a Journal© file are cropped so that they contain only information about the strokes in a single diagram and saved as ink.gif files using the Data Collector utility. Next, the InkConverter and drsXmlConverter are used to convert the ink.gif files to drs files and then the drs files to xml files. With xml files that contain single diagrams, the labeler, seen in Figure 3, is used to label each group of strokes that compose a symbol by hand. In the labeler utility, groups of strokes and substrokes can be circled using the digitizing pen and labeled by selecting the appropriate label, such as "and gate", from a pull-down menu. In Figure 3, the bottom wire has just been circled and the "wire(s)" option has been selected.

It is important to note that we consider a wire symbol to be the entire path from one end point to the other even if it is not straight or includes a jog.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <sketch id="99da93b4-f725-4c5f-bdfe-974fa44b1e8d">
  - <sketcher>
      <id>1013-1-25-06-n-p8-labeled</id>
    </sketcher>
    <study>E85</study>
    <domain>DigitalLogicDiagrams</domain>
    <point id="c3bd436e-292d-42f9-a141-e7a09b6cc81d" name="point" time="1137242878603" x="23.0"
      y="65.0" />
    <point id="d8957ce0-fa95-411d-8864-7734e1593bc4" name="point" time="1137242878678" x="24.0"
      y="65.0" />
    ...
  - <shape id="8e346239-24da-4ce3-947e-8e8153605441" laysInk="true" name="stroke"
      orientation="0.0" time="1143617165906" type="Stroke">
      <arg type="Point">258c3598-3e4e-41f5-901f-793ca85388b2</arg>
      <arg type="Point">cc62a5af-75fe-4874-8de2-34dbfb9710cf</arg>
      <arg type="Point">c68f0fde-94f0-4840-bfbf-a8aa964b9a35</arg>
      ...
    </shape>
    ...
  - <shape color="-16711681" id="6cda0423-2fef-4541-bd9c-d94453e0ef31" name="or" orientation="0.0"
      source="hand-labeled" time="1143617165968" type="or">
      <arg type="Stroke">f11dcb00-89c5-411e-a8af-b000f692c7fe</arg>
      <arg type="Stroke">805b990a-05a2-4646-9f05-329c92699ad7</arg>
    </shape>
    ...
  </sketch>
```

Figure 4: Portions of a Labeled xml File

The results of these conversions are labeled xml files which contain the necessary information about strokes in an easily accessible format. Figure 4 shows the general format of a labeled xml file. Toward the top of a file there is a list of point objects representing all the points in the diagram. These point objects store unique ids as well as the time they were drawn and the x and y coordinates of the point in the diagram. The time that each point was drawn is extrapolated using the time at which the stroke containing the point was drawn, when that point was drawn within the stroke (i.e. how many points have been drawn in the stroke up until now), and the sampling rate of the digitizing pen (i.e. the frequency at which points are created). Shape objects in labeled xml can be categorized as either strokes, substrokes, or symbols. As can be seen in Figure 4, each object with stroke information contains a unique id, the time the stroke was drawn, and a list of the point ids of points that are contained in that stroke. Objects containing substroke information contain similar fields, but they are different from objects storing stroke information because they are created in the labeling process when part of a symbol is drawn using a stroke that composes part of another symbol as well. In these cases, a substroke is created from the points that are in

the symbol being labeled. Substroke shape objects each store an additional "subStrokeOf" field that contains the id of the stroke with which it shares points. Finally, there are shape objects which store information about individual symbols. These objects contain the color of the symbol when it is displayed, a unique id, the type of symbol it was labeled as (in this case "or"), the time that the last stroke in the symbol was drawn, and a list of ids of the strokes and substrokes contained in the symbol. Within all the point and shape objects in a labeled xml file, the information about when strokes are created, where they are located, and what symbols they compose is stored.

After we create labeled xml files for the diagrams, we use an automated tool that we built to analyze the information that is available in the shape objects of the xml files. This tool scans each xml file line-by-line three times. The first time the xml file is viewed, the number of each type of shape object is recorded: stroke shape objects, substroke shape objects, and symbol shape objects. The second time the xml file is viewed, the id of each shape, the time it was drawn, and the label, are recorded along with the number of strokes in each shape. The final time the xml file is scanned, the ids of each stroke in symbol shape objects are recorded.

Once this information has been collected from the labeled xml files, our utility finds how many gates are drawn with non-consecutive strokes. It examines each gate, recording each stroke that is not contained in the gate, but was drawn at a time that falls in between the time that the first stroke in the gate was drawn and the time that the last stroke in the gate was drawn. We record all such strokes that were drawn during the time that a gate was drawn but are not part of the gate. We find out how often gates are comprised of a single stroke by checking the number of strokes in each gate shape object. Similarly, we find out how often wires are comprised of more than one stroke by checking the number of strokes in each wire shape object.

2.4 Accessing Information About the Order of Input and Output Wires Relative to Gates

As discussed above, answering the question about the order in which wires are drawn requires a different kind of labeling. In order to assess when gates are drawn relative to their input and output wires, the wires adjacent to gate symbols must be labeled according to their relationship with the gates to which they connect. Rather than designing a different, perhaps more complicated labeling system than that used in the previous section, we take advantage of the natural ability of humans to label symbols visually. We use the Ink Player utility,

also written by [7], to load the diagrams after they have been converted to ink.gif format. This utility allows the user to play the diagram back, stroke by stroke, in the order in which strokes were drawn. We play back the strokes in each diagram and take note of when each gate was drawn relative to the wires that connects to it.

2.5 Identifying Symbols with High Ink Density

In order to answer the question about ink density, the ink density for groups of strokes that do not compose symbols must be computed in addition to the ink density of groups that do compose symbols. The ink density of a group of strokes is defined by the percentage of space in a tight bounding box around the group that is covered by the strokes. In [4], Gennari *et al.* propose ink density as a useful statistic in the segmentation of analogue circuits. It is possible that because both analogue circuits and digital logic diagrams are comprised of two-dimensional shapes connected by a series of predominantly one-dimensional wires, ink density may be useful in segmenting gates in digital logic diagrams. Because the stroke segments in wires are often longer than those in gates, when wires are added to a group of strokes composing a gate, the size of the bounding box may increase, while the percentage of the bounding box that is covered by the strokes in the group drops.

Our method for calculating ink density is inspired largely by the method used by Gennari *et al.* We gain insight into the relevance of ink density to the segmentation of digital logic diagrams by first identifying groups of strokes with high ink densities and then producing a consistent mapping of these symbol candidates to the diagram, such that there are no overlapping symbols candidates.

For a given group of strokes, ink density is calculated through a series of steps. First, a tight bounding box is fit to the group. Because a system using the tightest bounding box would find symbol groups that compose straight lines, which are not gates, the larger of two tight bounding boxes taken at 45° angles from each other is used. Next, *hidden ink*, ink that is used as an approximation of the strokes of the pen when the pen is not touching the tablet surface, is added to the group of strokes. More precisely, a stroke of hidden ink is the stroke that is implicitly drawn between the end point of one actual stroke and the start point of the subsequently drawn stroke. Hidden ink is added because gates are often drawn as an outline of a shape, leaving the middle of the shape without any strokes. Adding hidden ink into the calculation of ink density tends to increase the density of gates. Once the hidden ink is added, the rectangle within the bounding box is rendered to a bitmap image and the percentage of pixels that contain ink is returned as the ink density.

In identifying groups of strokes with high ink density, we make three assumptions about the way in which the user draws that affect how we interpret the results. Firstly, we assume that the user draws gates using consecutive strokes. The validity of this assumption will be discussed in section 3.1, which discusses the results of each question. Secondly, our method assumes that each gate is drawn using more than one stroke. The validity of this assumption will also be discussed in section 3.2. The final assumption is that each gate is drawn using strokes that are completely contained in the gate. There are three diagrams in our data-set in which a single stroke is part of a wire and a gate. In these cases, we manually broke the stroke in question into two consecutive substrokes so that the part of the stroke in the gate could be considered separate from the part in the wire.

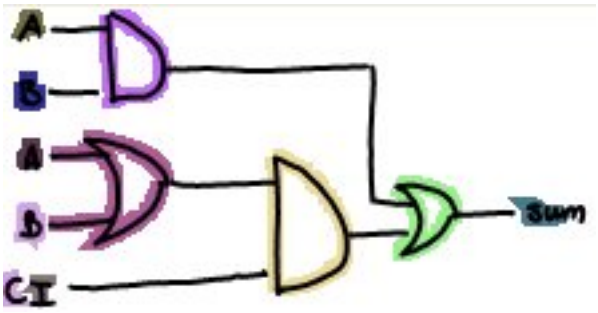

Figure 5: A Sample Segmentation Based on Ink Density

Given these assumptions, we identify the beginning stroke and the ending stroke in groups of strokes with high ink density through the following steps. First, the best ending strokes are found by considering each stroke in the diagram to be a potential beginning stroke and picking the stroke for which adding the next chronological stroke to the group decreases the ink density by more than 20%. Next, the beginning stroke that yields the highest ink density for each of the ending strokes found is selected. Starting with each ending stroke, consecutive former strokes are added until the ink density of the group of strokes drops. Once there is a list of groups of symbols with high ink density specified by beginning and ending strokes, overlapping groups are removed, giving preference to groups with higher ink density. The result is a consistent list of groups of strokes with high ink density. This list is displayed graphically using a different color to denote each high density group as in Figure 5. In this example, each gate is found correctly except the OR gate on the left side of the diagram which was found in a group with its input wires. In addition, this example shows that many groups of strokes forming text were found to have high density. For more detail about the method of ink density calculation used in this paper see [2].

3 Results and Discussion

Applying the methods of analysis described above to our data, we arrive at answers to the five research questions. The answers to these questions aid in designing a recognition system because they inform whether or not certain assumptions can be made about how digital logic diagrams are drawn. Table 2 shows how many gates and diagrams each subject drew. Since we gave the subjects no instructions on how many diagrams to draw or how to draw them, the number of diagrams each subject drew ranges from one to seven and the number of gates each subject drew ranges from two to twenty-four.

Table 2: Summary of Our Data

Subject	# of Diagrams	# of Gates
1	7	20
2	3	13
3	4	12
4	1	2
5	2	10
6	2	8
7	1	6
8	3	12
9	2	9
10	7	24
Total	32	116

3.1 The Order of Strokes in Gates

The results regarding the order of strokes composing gates suggest that assuming that gates are drawn using consecutive strokes is a valid assumption for the majority of gates, but that a significant portion of the subjects drew at least one gate using non-consecutive strokes. Among the 32 diagrams in the data-set, 13% (4) include gates with one or more stroke that was not drawn consecutively. In terms of quantities of gates, out of the 116 total gates in the data-set, 5% (6) were drawn with non-consecutive strokes. Although it is clear that gates are drawn using consecutive strokes the majority of the time, since many of the subjects (3 of the 10 subjects) drew at least one gate using non-consecutive strokes, it appears that it is not uncommon for a user to draw the occasional gate using non-consecutive strokes.

Further examination of the diagrams in which gates with non-consecutive strokes occur, shows that every gate with non-consecutive strokes was drawn such that the majority of the gate was drawn using consecutive strokes. However, a single stroke was added to the gate after the subject drew an average of 10 stokes while adding a label or drawing other parts of the diagram. The single, non-consecutive stroke in each case appears to have been added to correct a part of the gate that was not fully connected when the gate was drawn, or, simply to trace over an existing part of the gate, perhaps while the user contemplated what to draw next. A robust recognition system may assume that most gates are drawn using consecutive strokes, but take into account the fact that some gates may not be drawn in this manner by assuming that gates containing non-consecutive strokes often contain only a single one that spatially overlaps at least part of the consecutively drawn strokes.

3.2 The Number of Strokes in Gates

The results regarding the average number of strokes in each type of gate, shown in Table 3, suggest that it would be risky to create a recognition system that relies on the assumption that gates are drawn using more than one stroke, but that certain types of gates are more prone to being drawn in a single stroke. AND gates, which are the most commonly drawn type of gate, are frequently drawn in a single stroke. The average AND gate in our data-set is drawn in just 1.7 strokes; 38% of all the AND gates in the data-set were drawn in one stroke. The average number of strokes in OR gates, the second most frequent type of gate in our data-set, is 2.3 and thus greater than 2. However, as evidenced by the high standard deviation, many OR gates were drawn in one stroke; 33% of the OR gates in the data-set were drawn in one stroke.

While several AND and OR gates were drawn in single strokes, no NOT gates were drawn in fewer than two strokes, perhaps because the form of NOT gates, as defined in Figure 1, consists of two separate, adjacent shapes, a triangle and a circle. Similarly, XOR gates were never drawn in one stroke, likely because the shape of the XOR gate includes an arc that is not connected to the rest of the gate. Although NAND and NOR gates were not drawn enough times to state the average number of strokes in each type of gate with any certainty, it is likely that because the forms of these gates include circles similar to the one in the form of a NOT gate, NAND and NOR gates are usually drawn in more than one stroke.

The final type of gate, referred to as "unknown" in Table 3, is the non-standard gate shown in Figure 6. While this gate is not a standard gate, it appears once in each of two diagrams drawn by different subjects in their class notes during the same class session.

Table 3: Statistics Concerning the Number of Strokes in Each Type of Gate and Wires

	and	or	not	xor	nand	nor	unknown	wire	connector
total gates	58	24	16	11	4	1	2	288	8
total strokes	105	60	35	35	13	3	11	338	8
ave. strokes/gate	1.7	2.3	2.2	2.7	3.3	3	5.5	1.1	1
st. dev.	0.8	1.8	0.4	1.3	0.5	0	0.7	0.8	0

It is likely that the subjects were copying a diagram drawn by the professor as part of a lecture. The occurrence of this non-standard gate raises an important point, which is that a recognition system used in the classroom setting would ideally be versatile enough to attempt to recognize unknown symbols as *unknown* rather than labeling them incorrectly. In the domain of digital logic diagrams, it is possible that the number of strokes in unknown symbols could be useful in labeling them as unknown because, as in this case, the average number of strokes in the unknown symbol is noticeably higher than the average number of strokes in the known gates. Similarly, a recognition system could take advantage of the fact that only AND and OR gates are drawn using a single stroke, because if a single stroke is found to form a symbol during segmentation, and other information indicates that it is unlikely that the symbol is a wire, the label can be narrowed down to either AND or OR gate.

Figure 6: The Unknown Gate

3.3 The Number of Strokes in Wires

The results regarding the number of strokes in wires, also contained in Table 3, show that the average wire is comprised of roughly one stroke, but that an interesting pattern occurs when wires are drawn using more than one stroke. Only 10 % of wires contain more than

one stroke, and only 2% of wires contain more than two strokes. Therefore, a recognition system could take advantage of the assumption that wires are usually comprised of one stroke. However, it should be noted that in the cases in which wires are drawn in more than one stroke, recognition may be further complicated by the fact the wires are often comprised of strokes that were not drawn consecutively. While only 7% of the total wires in the data-set were drawn using non-consecutive strokes, 71% of the wires drawn using more than one stroke were also drawn using non-consecutive strokes. One explanation for the high percentage of multiple-stroke wires that contain non-consecutive strokes is that it is common for subjects to draw part of a wire as output from a gate, draw other parts of the diagram, and only finish drawing the wire once they are ready to draw the gate to which the wire inputs. It is possible that a recognition system could take this situation into account by looking for two non-consecutive strokes that each are linear, spatially close to each other on one of their ends, and spatially close to a gate on their other end.

3.4 The Order of Input and Output Wires Relative to Gates

The results for the question concerning the order in which input and output gates were drawn relative to the gate to which they connect, shown in Table 4, give a greater picture of the variety of ways in which wires are drawn. They show that it is not safe to assume that users draw diagrams in order from input to output. In order to assess when gates are drawn relative to the input and output wires that connect to them, we break down the possible cases. Each gate can either be begun before, or after the wires that serve as inputs to them. Similarly, each gate can either be begun before, or after the wire that serves as an output to it. For each gate we record when the input wires were drawn and when the output wires were drawn. As can be seen from Table 4, in 60% of the gates, the subject draws the input wires, then some part, if not all of, the gate, and then the output wire.

However, for 36% of the gates, the subjects drew one or more input wire after beginning the gate. In these cases, subjects often drew several of the gates first, before drawing any wires, perhaps to sketch the basic function of the diagram before filling in the details. Another interesting result is that it was more common for subjects to draw in reverse logical order, drawing the output first, then beginning the gate, and then drawing inputs, than it was for subjects to draw the inputs and the output before beginning the gate (4 vs. 1). This result seems fitting since if a subject draws in reverse logical order, he or she is still drawing wires and gates one after the other rather than drawing all connecting wires to a gate before drawing the gate.

Table 4: When Gates Were Drawn Relative to Adjacent Wires

Conditions
A: Gate is begun after all inputs and before output
B: Gate is begun before at least one input and before output
C: Gate is begun after all inputs and after output
D: Gate is begun before at least one input and after output

	A	B	C	D
number of gates	69	42	1	4
% of gates	60	36	1	3

3.5 The Ink Density of Gates

Our results regarding ink density, shown in Table 5, indicate that segmenting using ink density is not completely reliable. Given that the goal of segmentation using high ink density is to segment groups of strokes comprising gates, of the 116 gates in our data-set, only 33% were segmented correctly. 25% of the gates were found to be in groups with high ink density only when those groups included a section of wire. Many of these mis-segmented groups contain strokes that form a gate as well as relatively short strokes that form the inputs to that gate such as the OR gate in Figure 7(a). 11% of the gates were segmented incorrectly becuase only part of the gate was segmented. This type of mis-segmentation occurs mostly with gates that have sections that add area to the bounding box of the gate without adding a compensating amount of pixels containing ink, such as the NOT gate in Figure 7(b). The triangle part of the gate has a higher ink density than the whole gate including the circle. This type of mis-segmentation also occurs with NOR and NAND gates which have circles, and with XOR gates which have an extra arc that is not attached to the rest of the gate. While these features of the NOT, NAND, NOR, and XOR gates may be beneficial to an assumption based on the number of strokes per gate, as discussed above, they are not beneficial to an assumption based on the high ink density of the whole gate.

Although the rate of successful segmentations does seem not high, we have not taken into account the fact that the assumptions we made in designing our system (as described in Section 2.5) were not met in many cases. 6 gates are comprised of non-consecutive strokes. It is possible that these gates were mis-segmented because of this assumption. If these 6 gates were drawn using consecutive strokes, and were segmented correctly, the percentage of gates segmented correctly would increase to 44%. In addition, our segmentation system assumes that each gate is comprised of more than one stroke. To test if this assumption affects

Table 5: Results of Segmentation Based on Ink Density

	correctly segmented	missed completely	with wires	missing part	extra wires and missing part	with other gates	with other gates and wires
#	38	13	29	11	14	4 (2 groups)	7 (3 groups)
%	33	11	25	10	12	3	6

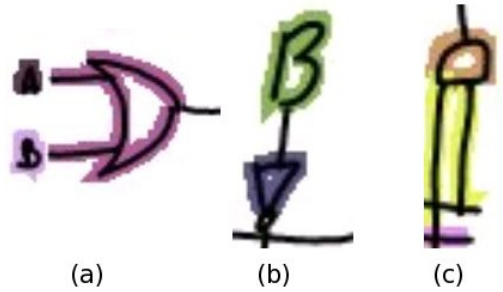

Figure 7: Examples of Mis-segmented Gates

segmentation, we manually split all 30 of the gates drawn in one stroke into two strokes so that the gates were comprised of two consecutive strokes. When we performed segmentation on those gates after making this change, 25 of these gates were segmented correctly. Figure 8 shows an example in which a gate that was missed initially, when it was comprised of one stroke, is segmented correctly once it is split into two strokes. If we take into account the correct segmentation of these 25 gates, 63% of all gates were segmented correctly. If we take into account both the 25 gates that would be segmented correctly if they were comprised of two strokes, as well as the 6 gates that may have been segmented correctly if they were comprised of consecutive strokes, 69% of the gates (63) would be segmented correctly. Although this percentage does not indicate a perfect segmentation, it shows that gates do have a tendency to have high ink densities. Unfortunately, groups of wires, such as the parallel wires in Figure 7(c), also have high ink densities; compared to the 38 to 63 gates that were segmented correctly, 29 groups included only wires. These results also show that using ink density to segment the gates in diagrams is much more effective when it is not based on the assumption that gates will be drawn in a single stroke. One additional result is that segmentation of gates using ink density would likely have to be performed only if all the text in the diagram has been removed; of the 500 or so groups found with high ink

density in our data-set, roughly 75% contained only text.

Figure 8: Example of Difference in Segmentation When Gates Have Multiple Strokes

4 Conclusion

We have examined five research questions that relate to the manner in which digital logic diagrams are drawn by users in the course of their everyday work. Possible questions for further research include: does pen speed differ between drawing wires and drawing gates, what is the range in the number of primitive components that compose a wire (this depends primarily on the number of jogs and arcs), and is there a common orientation for digital logic diagrams (e.g. do people usually draw the input side of gates on the left and the output side on the right)?

In this project, we learned valuable information about the ways users draw: they often draw gates using only consecutive strokes; they draw gates in a single stroke a significant amount of the time; they usually draw wires in single strokes, but when they do not, they often draw them using non-consecutive strokes; they do not always draw in the order of input wires, gate, and then output wire, in fact they draw the input wires to a gate after they begin drawing the gate over a third of the time; finally, users do not draw gates with significantly higher ink density than groups of strokes that contain wires. Overall, these results show that, while there is variation in the ways in which people draw digital logic diagrams, there are certain trends that come through. The trends found in this paper can be used in conjunction with others that may emerge from future research in the creation of a recognition system.

5 Acknowledgments

This work was made possible by the generosity of Michael Oltmans and Aaron Adler from the Computer Science and Artificial Intelligence Laboratory at MIT. They provided not only access to invaluable code but also helpful advice. This work was also greatly accelerated by

the work of Casey Chesnut. Finally, the inclusion of ink density calculations was made possible by the work of Dan Barcay.

References

[1] C. Alvarado and R. Davis. Sketchread: A multi-domain sketch recognition engine. *Proceedings of UIST 2004*, 2004.

[2] D. Barcay and M. Lazzareschi. Digital logic diagram segmentation by ink density. final project report for CS 153: Computer Vision, 2005.

[3] C. Chesnut. Converting journal notes to xml, svg, and onenote. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dntablet/html/tbconJournXML.asp, 2005.

[4] L. Gennari, L.B. Kara, T.F. Stahovich, and K. Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers and Graphics*, 29(4):547–562, 2005.

[5] S. Harris. Engineering 85: Digital electronics and computer engineering. http://www3.hmc.edu/ sharris/class/e85/, 2006.

[6] L.B. Kara and T.F. Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. *17th ACM User Interface Software Technology (UIST) 2004*, 2004.

[7] MIT Computer Science and Artificial Intelligence Laboratory. Rationale. http://www.rationale.csail.mit.edu/index.shtml.