Order this document by: AN1220/D

Optical Character Recognition Using Fuzzy Logic

by William A. Gowan

OVERVIEW

This application note shows how to envision, describe, and realize a design using fuzzy logic. It is not intended to be an introduction to fuzzy logic, but it is basic enough to be understood by designers with a cursory understanding of the subject. For those who seek an introduction to fuzzy logic, the Freescale Fuzzy Logic Educational Kit is an excellent primer. Other sources of information are shown in the list of references at the end of this document.

Fuzzy logic facilitates design of systems that mimic human reasoning. A fuzzy system accepts data input from sensors, then makes decisions based on that input. In most cases, these decisions are the basis for a control system. However, a fuzzy rule-driven system can simply be a classification engine that draws distinctions between and labels differing types of input data.

This note explains the design of an optical character recognition engine called the Optical Character Associator (OCA). Optical character recognition systems must classify optical inputs as specific letters, numbers, or other characters, and are thus ideal candidates for fuzzy logic implementation. OCA is a classification engine that recognizes the set of fourteen characters used by the US banking industry to encode account numbers along the lower edge of checks. The engine is implemented using an MC68HC11E9 8-bit microcontroller, although it could have been implemented using devices from the M68HC05, M68HC16, or M68300 MCU families.

OCA accepts input from a 64 x 1 pixel charge-coupled device (CCD) sensor. After an input preprocessor program formats sensor data into an easily "fuzzifiable" structure, the inference engine fuzzifies the data, applies the fuzzy rule set, and generates an output that corresponds to the character being read.

This application note presents OCA design methodology, and defines all input variables, fuzzy rules and output variables. Although preprocessor operation is fully described, and internal variables used by the preprocessor are explicitly defined, a designer must provide the actual preprocessing code in order to implement the system described. System resources not directly related to the optical portion of the system, such as motor transport for document movement beneath an optical sensor, must also be provided.

OCA was designed from simulated sensor data input. In order to implement a physical system, the simulated data should be carefully compared to actual character reads from the sensor to be used. Modification to fuzzy membership functions and rules may be required. The operation of OCA was verified using simulated testing as described in **TESTING RESULTS**. Freescale does not guarantee the operation of the software described in this document.

PROBLEM DEFINITION

The industry definition for the character set to be recognized appears in **Figure 1**. There are fourteen valid characters — numeric characters zero through nine, and four special characters, SS1 through SS4.

freescale™
semiconductor

Freescale Semiconductor, Inc.

AN1220
OCR CHAR

Figure 1 Character Set To Be Recognized

Each character is right-justified in a 125 mil wide frame. Other characters cannot intrude into the frame. The widest characters, 0, 8, SS1, SS2, SS3, and SS4, have a specified width of 91 mils. Characters 4, 6, and 9 are specified at 78 mils wide. Characters 3, 5, and 7 are specified at 65 mils. Characters 1 and 2 are specified at 52 mils. Because of the differing specified character widths, there is a variable amount of white space to the left of each character in a string of characters.

The optical sensor chosen for this design is the Texas Instruments TSL214. This device consists of 64 vertically-aligned CCD elements. Each pixel is 4.72 mils wide and 2.756 mils high —a data "slice" is 4.72 mils wide and 319 mils high. Since maximum character height is 117 mils, there is a pixel area of approximately 100 mils above and below each character.

Spacing between data slices is determined by the relationship between the width of one data slice and the width of a character, or more specifically, the width of the narrowest line segment of a character.

The width of the narrowest line segment of a character (for instance, the thin vertical line that forms the left side of the character 0) is 13 mils. To insure detection, a minimum of two slices must be taken in the width dimension of any segment. If two slices were not taken, a line segment could straddle two data slices and thus not be detected. Data slices need not touch each other, but the gap between them must be small. Dividing a 125 mil frame into 22 vertical slices yields a spacing between data slices of 5.68 mils. This spacing insures that at least two samples are taken in a 13 mil wide character element.

The TSL214 has a specified integration time, measured in ms, which is a function of light intensity. Satisfying the integration time specification allows every CCD in the device to respond to the light level striking it. For light intensities ranging from 15 to 42 $\mu w/cm^2$, an integration time of 6 ms is adequate.

After integration time has elapsed, data may be read out of the optical sensor serially as analog values. When the sensor SI input is enabled, sensor output voltage represents the analog value from CCD#1. Upon the next clock transition, the output becomes the analog value from CCD#2, and so on until all 64 pixels have been read out.

Since sensor data is produced in the form of an analog value, an MCU A/D converter channel can be used to read the value in. In this high-contrast application, it is also possible for sensor output to be read as serial digital data, provided that saturated CCD output is greater than TTL V_{IH} and unsaturated output is less than TTL V_{IL}. Backlighting the document being scanned with a bright red LED provides high contrast.

At this point the problem can be defined. Hardware must provide a mechanism to light the document and move it under the sensor in 5.68 mil increments. The microcontroller receives a 64-bit stream of values for each slice. From this data, the classification engine must correlate contiguous data slices against the labels of recognizable characters.

AN1220/D

THE FUZZY LOGIC DESIGN PROCESS

The three elements required to realize a fuzzy system are fuzzification, rule application, and defuzzification. Fuzzification is the scaling of input data to the universe of discourse (the range of possible values assigned to fuzzy sets). Rule application is the evaluation of fuzzified input data against the fuzzy rules written specifically for the system. Defuzzification is the generation of a specific output value based on the rule strengths that emerge from the rule application.

In a realized fuzzy system, a microcontroller or other engine runs a linked section of object code that consists of two segments. One segment implements the fuzzy logic algorithm, performing fuzzification, rule evaluation, and defuzzification, and thus can be thought of as a generic fuzzy logic inference engine. The other segment ties the expected fuzzy logic inputs and outputs, as well as application-specific fuzzy rules, to the fuzzy logic inference engine.

A sophisticated development environment is required to generate microcontroller object code from input in the form of input variables, output variables, and fuzzy rules. Freescale currently provides two fuzzy logic development environments.

The Knowledge Base Generator, KBG11B.EXE, is a freeware development environment that supports a fuzzy inference engine called FUZZY11B.ASM. KBG11B.EXE runs under MS-DOS. It provides a graphical interface for the creation of input and output membership functions.

The Fuzzy Inference Development Environment (FIDE), by Aptronix, also runs on the IBM-PC platform, in the Microsoft Windows environment. FIDE offers an extensive graphical interface for development and debugging, as well as system-level simulation. OCA was developed with the Aptronix FIDE tool.

THE DATA PREPROCESSOR

As defined, the OCA problem presents certain obstacles that make pattern matching on a bit-by-bit basis impractical. For instance, the edge of a character segment can show up in two or more data slices, depending on where the slices overlap. Further, slight variations in printing cause character height and width to vary. Misfeeding of the document can skew the imaged character. Fuzzy logic, which is inherently superior for processing imprecise data, is a natural for this application. However, a data preprocessor is necessary to simplify the problem so that it can be easily described in fuzzy rules.

Figure 2 illustrates the difficulties a programmer encounters when trying to match incoming bit patterns against an idealized bit pattern, or template. Each of the three sections of Figure 2 shows nineteen data slices of typical reads of the character 0. Leading white space is not shown because this representation is left justified with respect to the first data slice that produces valid data.

The leftmost portion of **Figure 2** represents the bit pattern associated with an ideal read of a character 0. This portion of the figure can be considered to be a template for the read of a character 0. The center portion of **Figure 2** shows the bit pattern of a misaligned character, and the right portion of **Figure 2** shows the bit pattern of a skewed character.

For this discussion, consider a darkened pixel to be a bit with a logical value of 1.

One approach to recognition would have a program compare scanned characters to templates on a bit-for-bit basis. Clearly, this procedure could often fail. For instance, the program would expect a 1 in slice 1, bit 31 of a character 0, and neither misaligned nor skewed characters would satisfy the expectation.

Another approach would have the program sum all the bits in each slice and compare the resulting slice totals to corresponding slice totals from templates. As shown in **Table 1**, this approach can produce a match in the misaligned case. Unfortunately, it fails in the skewed case.

| IDEAL | MISALIGNED | SKEWED |
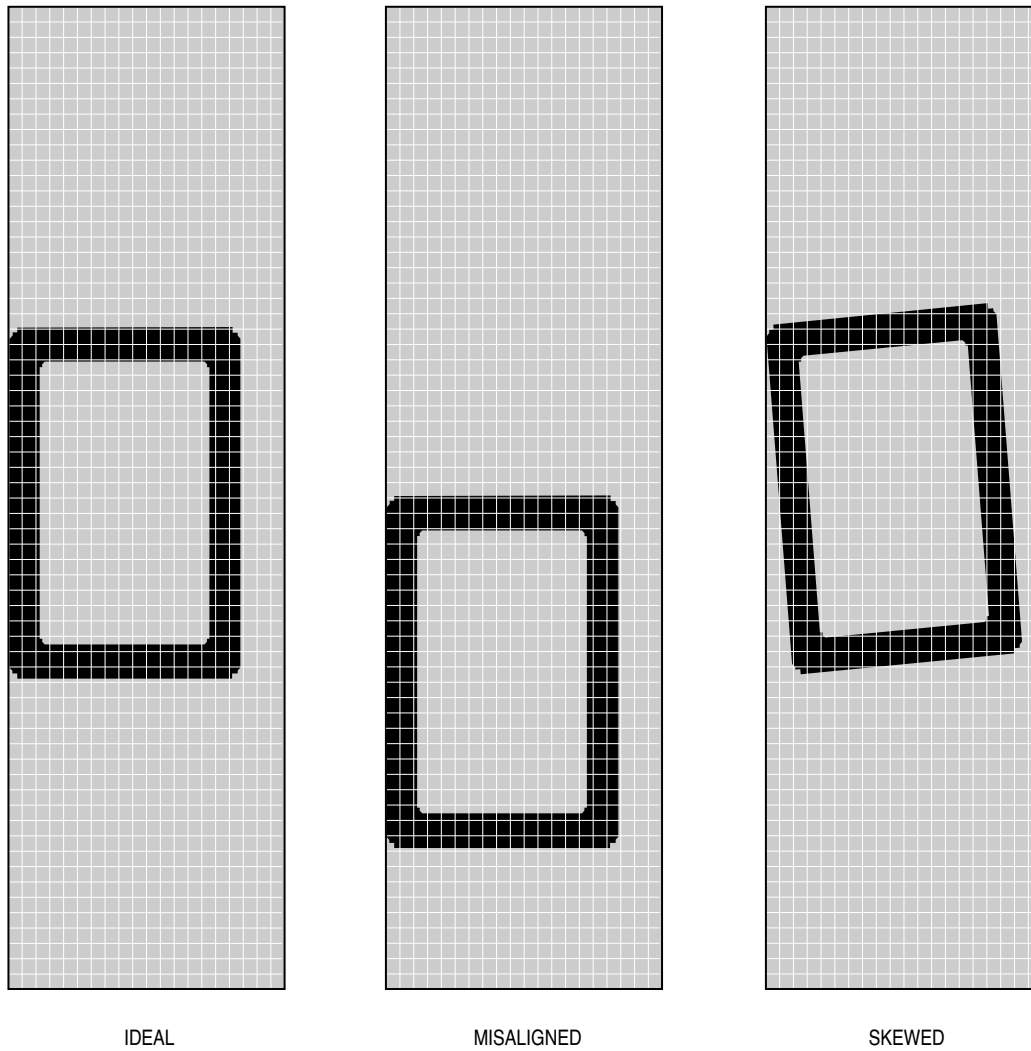
AN1220
OCR SLICE

Figure 2 19 Data Slices for Ideal, Misaligned, and Skewed Character 0

Table 1 Slice Totals for the Three Readings of Figure 2

Slice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ideal	23	23	5	4	4	4	4	4	4	4	4	4	4	4	23	23	0	0	0
Misaligned	23	23	5	4	4	4	4	4	4	4	4	4	4	4	23	23	0	0	0
Skewed	6	21	21	8	5	5	6	5	5	5	5	4	4	4	12	23	16	3	0

PREPROCESSOR OUTPUT: THE TRANSITION CONCEPT

The data in **Table 1** provides a useful insight. It is apparent in all three cases that the magnitude of the slice total increases to a high value of approximately 23, decreases to a low value of approximately 4, increases again to a high value of approximately 23, and then finally decreases to zero. **Figure 3** is a plot of the slice total for the three cases. Even though the bit patterns and slice totals are different, plots of the slice totals have the same shape.

AN1220/D

4

AN1220
OCR SLICE COMP
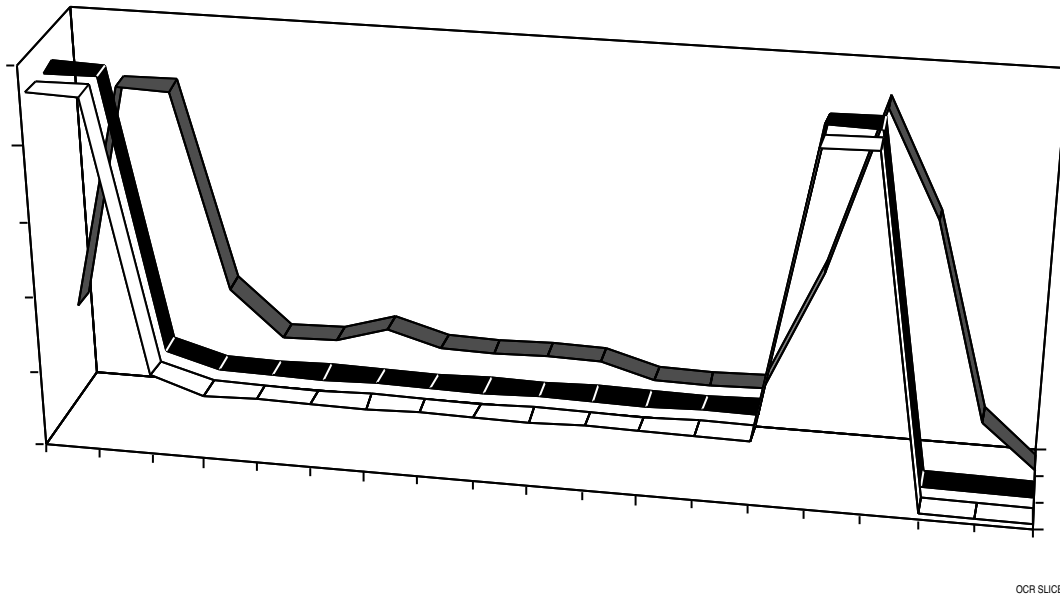
Figure 3 Graphical Comparison of Slice Totals from Figure 2

It is not difficult to write a program that locates and quantifies these transitions, or changes of magnitude, in slice totals. Quantified transitions will form the input to the OCA fuzzy engine. The fuzzy rules will look something like this: *A very large positive transition, followed by a large negative transition, followed by a large positive transition, followed by a very large negative transition, indicates a character zero*.

For this note, a transition is defined as the difference between a current local maximum (or minimum) and the previous local minimum (or maximum). The data preprocessor takes a data slice, obtains its slice total, and compares the magnitude of the slice total to previous slice totals to determine whether it constitutes a new local maximum or minimum.

Figure 4 shows the preprocessor algorithm that takes in slice data and generates transition outputs. Variables that are updated during preprocessor operation are listed in **Figure 4**. Preprocessor outputs take the form of a transition number and an associated transition magnitude. For instance, X1 = 23 means transition number 1 has a magnitude of 23.

Notice that the algorithm incorporates hysteresis in determining a direction change. In other words, a transition must be of three bits or greater magnitude to be recognized. For instance, if a current reading produces a slice total of 6 and the previous reading left DIR as –and I_min as 4, the current reading would fail the test CR > PR+2, but would pass the test CR > PR. Since DIR - +, none of the variables are changed.

The preprocessor algorithm has no effect on system throughput because it can be run during the delay for integration time.

Table 2 shows how variables are updated after each slice. The slice data applied is from the skewed case shown in **Figure 2**. Prior to entering the routine *Transition Calculation*, variables are initialized to the values shown in the column labeled Init. Data slice #1 is defined as the first slice with a slice total greater than 2. A final transition number/magnitude calculation is forced after the 22nd slice. **Figure 5** provides a graphical explanation of the preprocessor algorithm as applied during the calculation of **Table 2**.

AN1220/D

5

Freescale Semiconductor, Inc.
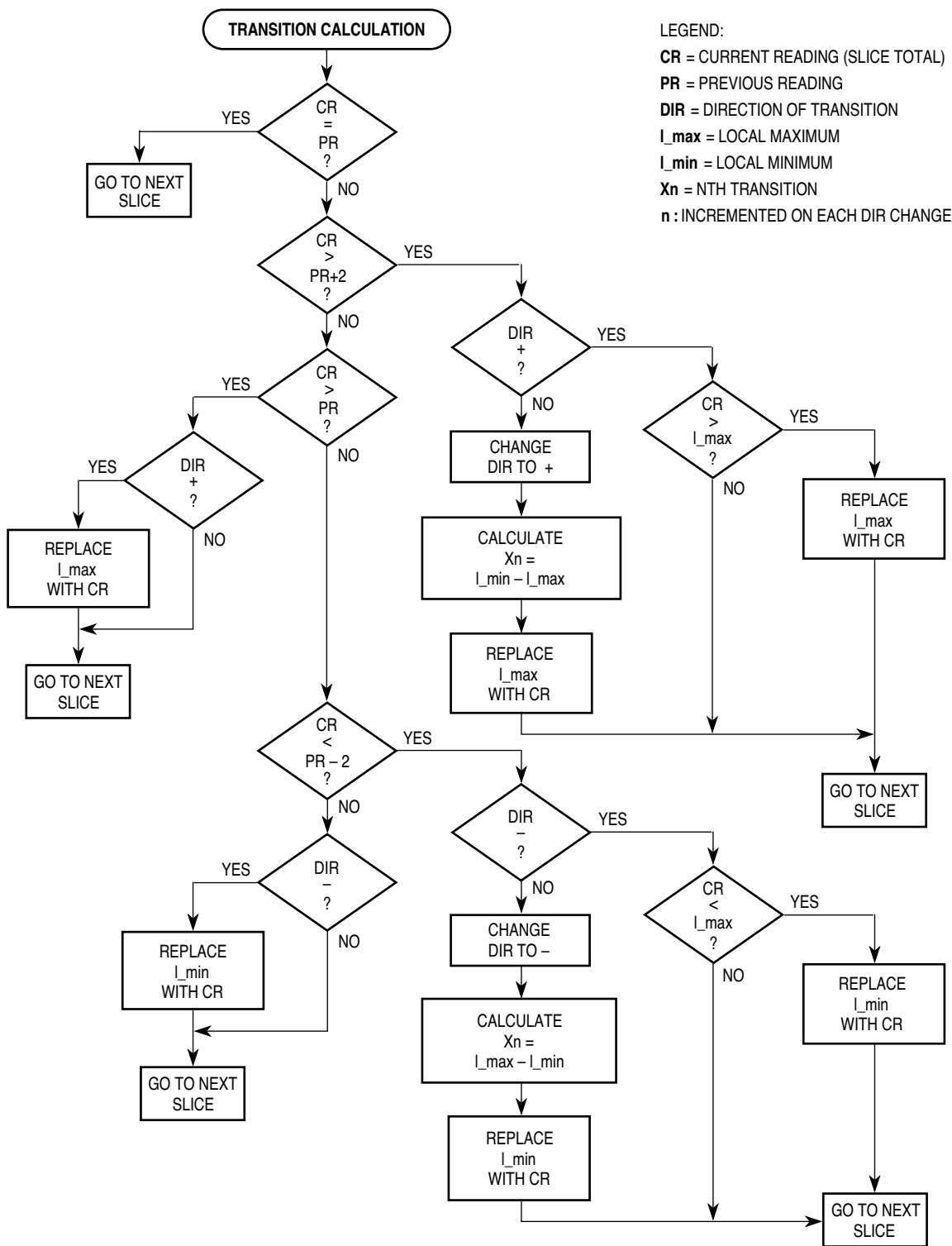
Figure 4 The Preprocessor for Transition Calculation

Freescale Semiconductor, Inc.

Table 2 Translation Calculation for Skewed Character Scan

Slice	init	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
CR	0	6	21	21	8	5	5	6	5	5	5	5	4	4	4	12	23	16	3	0	0	0	0
PR	0	0	6	21	21	8	5	5	6	5	5	5	5	4	4	4	12	23	16	3	0	0	0
Dir	+	+	+	+	−	−	−	−	−	−	−	−	−	−	−	+	+	−	−	−	−	−	−
X					X1											X2		X3					X4
I_min	0	0	0	0	8	5	5	5	5	5	5	5	4	4	4	4	4	16	3	0	0	0	0
I_max	0	6	21	21	21	21	21	21	21	21	21	21	21	21	21	12	23	23	23	23	23	23	23
X magnitude					21											−17		19					−23
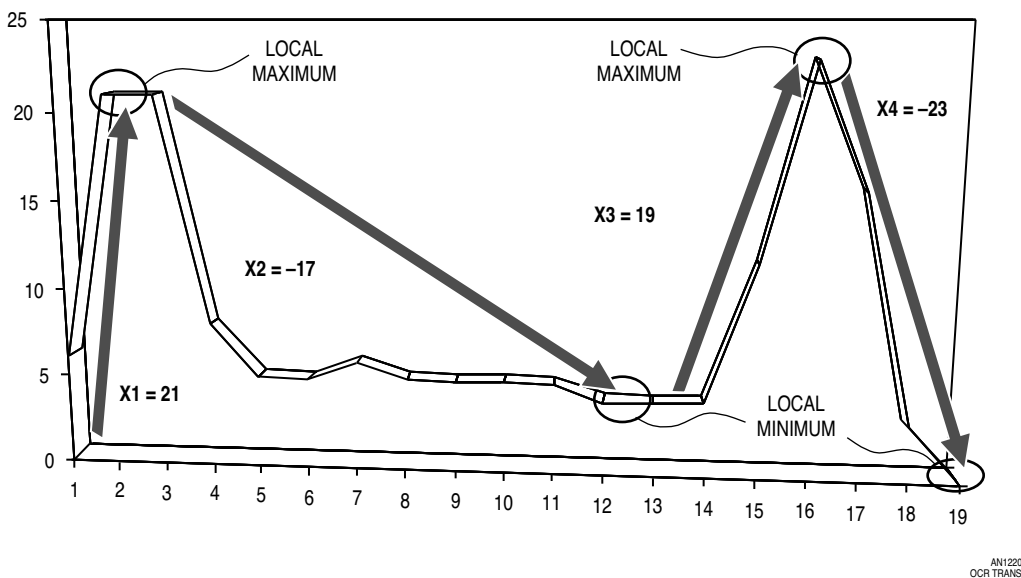
Figure 5 Visual Representation of Table 2 Transitions

The preprocessor found the following four transitions while reading this character.

$$X1 = 21$$
$$X2 = -17$$
$$X3 = 19$$
$$X4 = -23$$

For comparison, the preprocessor would return the following values for both an ideal and a misaligned character 0:

$$X1 = 23$$
$$X2 = -19$$
$$X3 = 19$$
$$X4 = -23$$

AN1220/D

7

FUZZIFYING TRANSITION INPUTS

The transitions visualized in Figure 5 make it very easy to write a fuzzy rule that recognizes a character 0: If X1 is Very_Large_Positive and X2 is Large_Negative and X3 is Large_Positive and X4 is Very_Large_Negative, then Char is 0.

Clearly, a similar visualization of all fourteen characters is required to write the remaining rules. **Table 3** presents the range of transition magnitudes for all fourteen characters. This data was obtained by simulating each character eight times, with skews up to 5% in each direction. Notice that the number of transitions per character varies from two (characters 1 and 3) to six (characters 6, SS1, SS2, and SS4).

Table 3 Transition Magnitude Ranges For All Fourteen Characters

Character	X1(lo,hi)		X2(lo,hi)		X3(lo,hi)		X4(lo,hi)		X5(lo,hi)		X6(lo,hi)	
0	21	24	−17	−19	16	20	−20	−24	:	:	:	:
1	22	24	−22	−24	:	:	:	:	:	:	:	:
2	15	16	−8	−8	7	8	−15	−16	:	:	:	:
3	22	24	−22	−24	:	:	:	:	:	:	:	:
4	18	19	−15	−17	7	9	−10	−11	:	:	:	:
5	14	16	−7	−9	8	9	−15	−16	:	:	:	:
6	23	24	−15	−18	3	6	−5	−11	5	6	−10	−11
7	7	8	−5	−6	11	13	−8	−10	4	5	−9	−10
8	20	23	−13	−17	15	17	−22	−23	:	:	:	:
9	13	13	−8	−9	18	20	−23	−24	:	:	:	:
SS1	10	11	−10	−10	10	11	−10	−11	10	11	−10	−11
SS2	16	17	−16	−17	15	17	−13	−17	9	11	−10	−11
SS3	15	16	−15	−16	16	16	−15	−16	:	:	:	:
SS4	8	8	−8	−8	8	8	−8	−8	8	8	−8	−8

The first step in fuzzifying this data is to establish a universe of discourse that defines the range of possible values for fuzzy inputs. Once the universe of discourse is defined, fuzzy sets can be created within it.

In this case, X1, X2, X3, X4, X5, and X6 are the fuzzy inputs. From Table 3, transition magnitudes, measured in pixels, vary from −24 to +24. Since a slightly oversized character or stray marks on the document can cause more pixels to be counted, the universe of discourse is represented by the range of values from −30 to +30 pixels.

Figure 6 shows the distribution of transition values across the universe of discourse. The labels denote each character and the transition number, followed by a graphical representation of that transition's range, from **Table 3**.

There are actually two universes of discourse: a positive one associated with X1, X3, and X5, and a negative one associated with X2, X4, and X6. Since transitions of 2 pixels or less are to be ignored, the positive universe of discourse is defined as the range of values from 2 to 30 pixels, and the negative universe of discourse is defined as the range of values from −30 to −2 pixels.
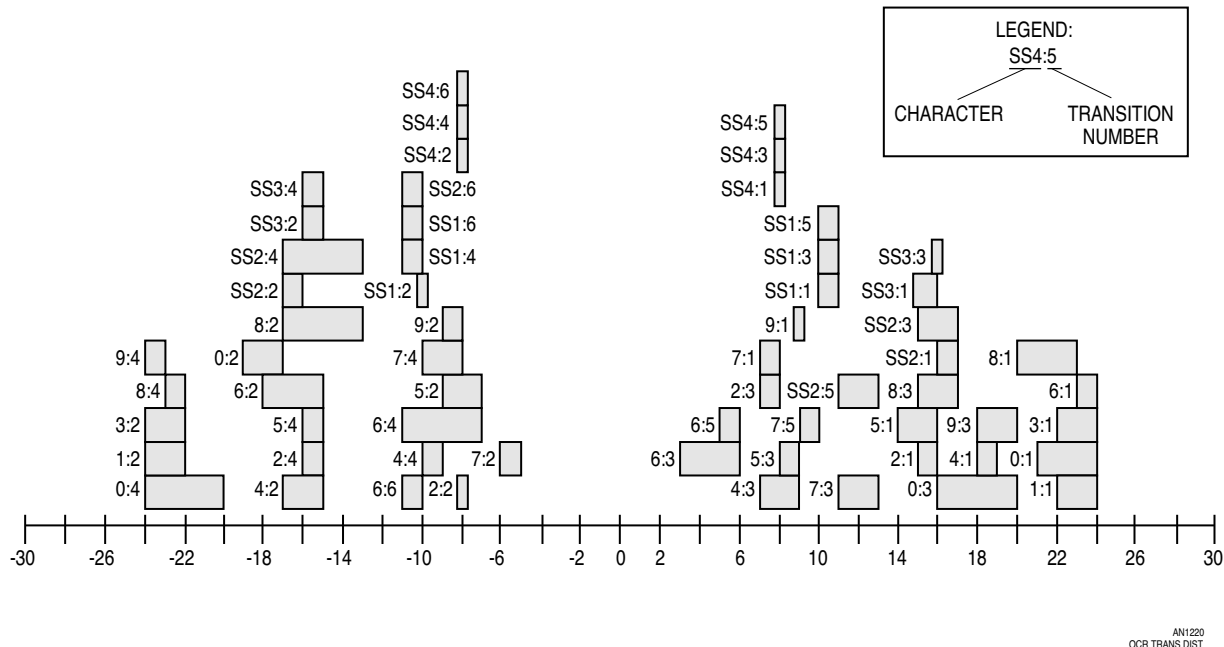
Figure 6 Distributions of Transitions Across the Universe of Discourse

It is obvious from Figure 6 that there is some clumping of transition data, especially on the negative side. **Figure 6** could be used to create fuzzy sets that apply to all six transition inputs. However, the fuzzy sets can contribute to more precise character recognition if they are made specific to each transition number.

Figure 7a, **Figure 7b**, and **Figure 7c** separate transition data by transition number. Based on the natural grouping of data, fuzzy sets are assigned and labeled.

Notice from **Figure 7c** that transitions 5 and 6 do not group well and thus do not contribute substantially to character discrimination. Therefore, fuzzy sets are not assigned to transition numbers X5 and X6, and these transitions do not appear as part of the fuzzy rules.

Another observation: negative transitions X2 and X4 fall into clearly delineated sets, but positive transitions X1 and X3 do not. The fuzzy set boundaries for transitions X1 and X3 are therefore chosen more arbitrarily. Several positive transitions have a degree of membership in more than one fuzzy set.

It is important to understand that a fuzzy input variable must generate at least some association with its fuzzy set across the entire range of possible values for that fuzzy input. For example, the range of values produced by character 4 in transition 1 is 18 –19. Despite this small range, all possible values of X1 returned by reads of character 4 satisfy neither fuzzy sets Med nor Max (the value 19 has an association of 0 with Med, and the value 18 has an association of 0 with Max). Thus, the fuzzy set X1 Large is added to properly classify character 4. Likewise, fuzzy set X3 Large is added for character 7.

At this point, transition inputs and fuzzy sets are defined. Unfortunately, transition inputs alone are not adequate to classify characters. Consider the characters 1 and 3. Each has only two transitions, X1 and X2. The rule for character 1 is: If X1 is Max and X2 is Large then Character is 1. The rule for character 3 is: If X1 is Max and X2 is Large then Character is 3. Thus, from transition data, 1 and 3 are indistinguishable.

Table 4 shows the fuzzy magnitude of each transition presented by character. The Conflicts column shows which other characters present the same fuzzy magnitude. Several conflicts occur, so it is necessary to introduce input variables in addition to transition magnitude.
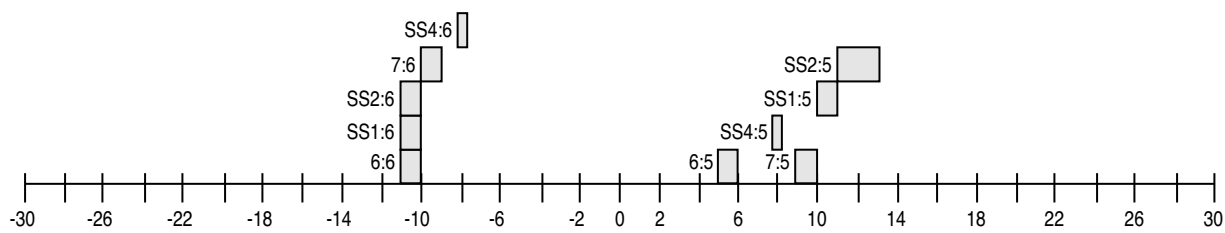
AN1220/D

9
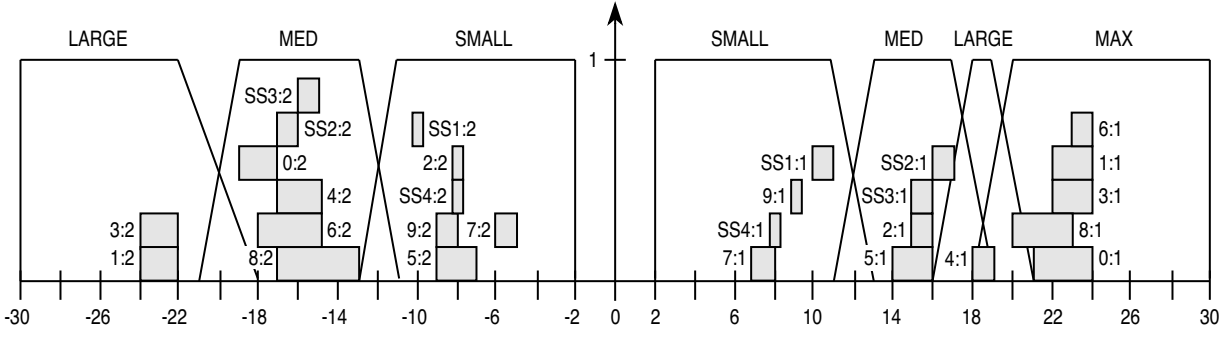
Figure 7a X2 (Left) and X1 (Right) Fuzzy Set Definitions
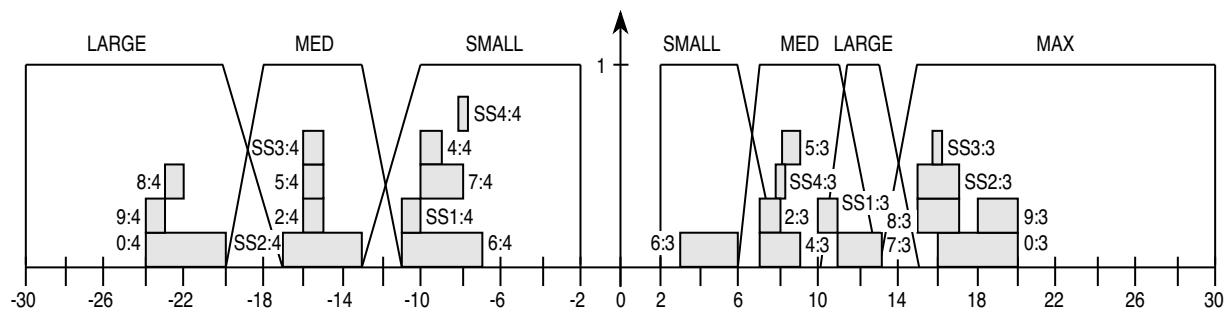
Figure 7b X4 (Left) and X3 (Right) Fuzzy Set Definitions

Figure 7c X6 (Left) and X5 (Right) Transition Distributions

AN1220/D

10

**Table 4 Transition Magnitudes Presented By Each Characte**

Character	X1	X2	X3	X4	Conflicts
0	Max	Med	Max	Large	8
1	Max	Large	:	:	3
2	Med	Small	Med	Med	5
3	Max	Large	:	:	1
4	Large	Med	Med	Small	:
5	Med	Small	Med	Med	2
6	Max	Med	Small	Small	:
7	Small	Small	Large	Small	:
8	Max	Med	Max	Large	:
9	Small	Small	Max	Large	:
SS1	Small	Small	Med	Small	SS4
SS2	Med	Med	Max	Med	SS3
SS3	Med	Med	Max	Med	SS2
SS4	Small	Small	Med	Small	SS1

PREPROCESSOR OUTPUT: SUM-OF-PIXELS (SOP)

Some of the conflicts shown in **Table 4** can be resolved by considering the total dark area in each character image. Total dark area is measured as a sum of pixels, or SOP, and can easily be summed during the operation of the transition calculation preprocessor. **Table 5** shows the range of SOP values for each character. **Figure 8** shows the universe of discourse and fuzzy sets for SOP.

Table 5 SOP Range for Each Character

Character:	0	1	2	3	4	5	6	7	8	9	SS1	SS2	SS3	SS4
SOP (lo)	139	112	99	132	124	110	138	79	194	149	102	140	177	84
SOP (hi)	167	130	113	146	142	125	174	91	201	159	123	158	190	88
SOP (avg)	153	121	107	137	135	119	150	84	197	154	111	148	182	86
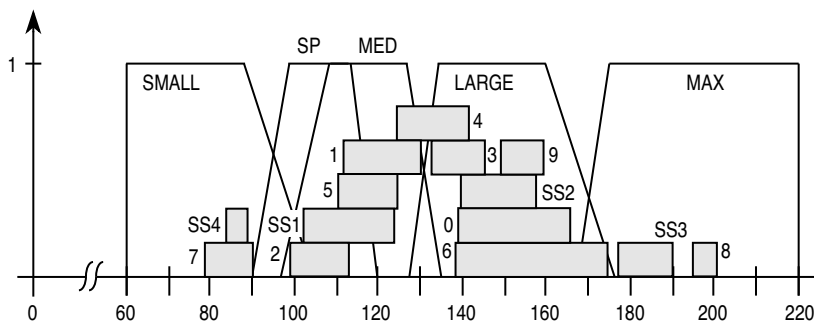
Figure 8 SOP Fuzzy Set Distributions

AN1220/D

Figure 8 shows how conflicts shown in **Table 4** are resolved by the addition of the SOP input variable. Consider the conflict between characters 0 and 8. As long as the possible range of SOP values for 0 always produces a higher degree of membership in the fuzzy set Large than in the fuzzy set Max, a 0 is recognized as a 0 rather than as an 8. SOP for character 0 ranges from 139 to 167 (Table 5). Figure 8 shows that any value of SOP from 133 to 160 produces a degree of membership of 1 in the fuzzy set SOP Large. Values from 161 to 176 produce declining degrees of membership. Since the maximum value of SOP from character 0 is 167, the character 0 always produces some degree of membership in the fuzzy set SOP Large and none in the fuzzy set SOP Max. Therefore, a 0 is never recognized as an 8. Conversely, the range of SOP values for the character 8 always produces a degree of membership of 1 in the fuzzy set SOP Max and a degree of membership of 0 in the fuzzy set SOP Large, so that an 8 is never recognized as a 0. The conflict is completely resolved. Each character appears exclusively within a fuzzy set with no overlap. The conflict between SS2 and SS3 is resolved in the same way.

Although there is some overlap, the conflict between SS1 and SS4 is also completely resolved. No value for SS4, which is a member of the fuzzy set SOP Small, will ever generate membership in the fuzzy set SOP Med associated with SS1. While very low values for SS1 can cause low degrees of membership in SOP Small, there will be a higher degree of membership in SOP Med and the correct selection will be made.

The conflict between 1 and 3 is resolved in a similar way. Low values for 3 generate some membership in SOP Med, but cause a higher degree of membership in SOP Large, so 3 is correctly recognized. Likewise, high values for 1 generate some degree of membership in SOP Large, but cause a greater degree of membership in SOP Med. Thus, a correct result is returned.

A special fuzzy set labeled Sp helps to resolve conflict between 2 and 5. Here, the SOP range for character 5 is assigned to the fuzzy set SOP Med, whereas for 2 it is assigned to SOP Sp. However, all values of SOP for 2 generate some degree of membership in SOP Med and only high values of SOP for 5 fail to generate membership in SOP Sp. Therefore, while SOP in most cases helps resolve the conflict between 2 and 5, an additional input variable is required.

Notice that the range of SOP values for character 4 is not adequately represented by either SOP Med or SOP Large. In this case, it is best to leave the SOP input variable out of the fuzzy equation for character 4.

PREPROCESSOR OUTPUTS: TERMINATION WIDTH (TERM)

An additional characteristic that may resolve the remaining conflict is character width, measured in data slices. Character width is called TERM, a designator for the terminating data slice. TERM is easy to determine during the preprocessor phase —it is simply the data slice number of the last non-zero slice. **Table 6** shows the range of values for TERM for each character. **Figure 9** shows the universe of discourse and fuzzy sets for TERM. Figure 9 also shows that the conflict between 2 and 5 has been resolved. All TERM values returned by a read of character 2 fall in the Small set, while all values for 5 fall in the Med set.

TERM is required only to resolve the conflict between characters 2 and 5. However, it makes sense to assign the remaining characters to fuzzy sets in the TERM universe of discourse. The minimal additional code required for these rule additions produces better qualified results and a more robust classification system.

Because of the interaction of their TERM ranges with multiple fuzzy sets, TERM should not be used as a qualifier for characters 7 and 9. As Figure 9 shows, the largest value of TERM for 9 produces a higher degree of association with Max than with Large, which prevents OCA from generating the output 9. Likewise, the lowest value of TERM for 7 produces a higher degree of association with Small than with Med, preventing a resultant 7 for low values of TERM.

Table 6 TERM Range for Each Character

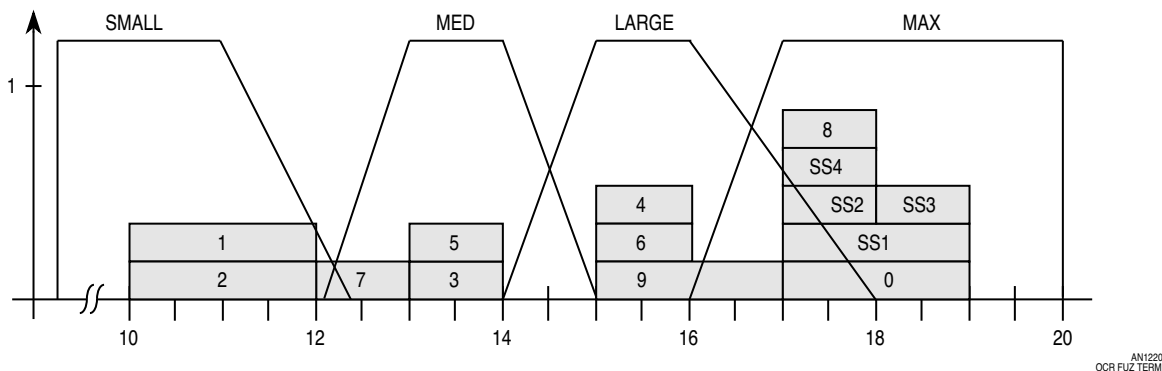| Character:	0	1	2	3	4	5	6	7	8	9	SS1	SS2	SS3	SS4
TERM (lo)	17	10	10	13	15	13	15	12	17	15	17	17	18	17
TERM (hi)	19	12	12	14	16	14	16	13	18	17	19	18	19	18

AN1220/D

12

Figure 9 TERM Fuzzy Set Definitions

CREATING FUZZY RULES

Fuzzy rules create associations between specific inputs and desired outputs. Here, the six input variables are X1, X2, X3, X4, SOP and TERM. Fourteen valid output variables, one for each of the fourteen characters to be recognized, are defined.

Table 7 shows the relationship between fuzzy input variables and fuzzy output variables.

Table 7 OCA Fuzzy Set Associations

Character	X1	X2	X3	X4	SOP	TERM
0	Max	Med	Max	Large	Large	Max
1	Max	Large	:	:	Med	Small
2	Med	Small	Med	Med	Sp	Small
3	Max	Large	:	:	Large	Med
4	Large	Med	Med	Small	:	Large
5	Med	Small	Med	Med	Med	Med
6	Max	Med	Small	Small	Large	Large
7	Small	Small	Large	Small	Small	:
8	Max	Med	Max	Large	Max	Max
9	Small	Small	Max	Large	Large	:
SS1	Small	Small	Med	Small	Med	Max
SS2	Med	Med	Max	Med	Large	Max
SS3	Med	Med	Max	Med	Max	Max
SS4	Small	Small	Med	Small	Small	Max

With the relationship information summarized in **Table 7**, writing the fuzzy rules becomes simple. The fuzzy rules appear at the end of the listing which follows.

AN1220/D

13

```
$fiu for an optical character associator

$This application accepts optical data which has been processed
$into five input data classifications:
$ X1 (Transition 1)
$ X2 (Transition 2)
$ X3 (Transition 3)
$ X4 (Transition 4)
$ SOP (Sum Of Pixels)
$ TERM (Data Slice Number corresponding to TERMination Length)

$Where applicable, this program classifies incoming data as one of
$fourteen possible characters, as defined by an industry standard
$specification for the printing of account numbers on bank checks.

$Created August 6, 1992
$Last modified: November 24, 1992

$Filename: OCA.FIL

fiu tvfi *8;

$Variable Definitions: X1, X2, X3, and X4 are defined as
$transitions; the magnitude of change between local minima and maxima.

invar X1 "delta pixels" :2 () 30 [
 Small (@2,1, @11,1, @13,0),
 Med (@11,0, @13,1, @17,1, @19,0),
 Large (@16,0, @18,1, @19,1, @21,0),
 Max (@18,0, @20,1, @30,1)];

invar X2 "delta pixels" :-30 () -2 [
 Large (@-30,1, @-22,1, @-18,0),
 Med (@-21,0, @-19,1, @-13,1, @-11,0),
 Small (@-13,0, @-11,1, @-2,1)];

invar X3 "delta pixels" :2 () 30 [
 Small (@2,1, @6,1, @8,0),
 Med (@6,0, @7,1, @11,1, @13,0),
 Large (@10,0, @11,1, @13,1, @15,0),
 Max (@13,0, @15,1, @30,1)];

invar X4 "delta pixels" :-30 () -2 [
 Large (@-30,1, @-20,1, @-17,0),
 Med (@-20,0, @-18,1, @-13,1, @-11,0),
 Small (@-13,0, @-10,1, @-2,1)];

invar SOP "total pixels" :60 () 220 [
 Small (@60,1, @88,1, @104,0),
 Sp (@90,0, @99,1, @113,1, @120,0),
 Med (@97,0, @109,1, @128,1, @134,0),
 Large (@126,0, @133,1, @160,1, @176,0),
 Max (@167,0, @178,1, @220,1)];

invar TERM "data slices" :8 () 20 [
 Small (@8,1, @11,1, @12.3,0),
 Med (@12.1,0, @13,1, @14,1, @15,0),
 Large (@14,0, @15,1, @16,1, @18,0),
 Max (@16,0, @17,1, @20,1)];
```

```
Outvar Char "Character" :0 () 14 * (
 one =1,
 two =2,
 three =3,
 four =4,
 five =5,
 six =6,
 seven =7,
 eight =8,
 nine =9,
 zero =10,
 SS1 =11,
 SS2 =12,
 SS3 =13,
 SS4 =14);


$!!! Fuzzy Rules !!!
$Defines relationships between input variables and output variables.

if X1 is Max and X2 is Med and X3 is Max and X4 is Large
 and SOP is Large and TERM is Max
 then Char is zero;

if X1 is Max and X2 is Large
 and SOP is Med and TERM is Small
 then Char is one;

if X1 is Med and X2 is Small and X3 is Med and X4 is Med
 and SOP is Sp and TERM is Small
 then Char is two;

if X1 is Max and X2 is Large
 and SOP is Large and TERM is Med
 then Char is three;

if X1 is Large and X2 is Med and X3 is Med and X4 is Small
 and TERM is Large
 then Char is four;

if X1 is Med and X2 is Small and X3 is Med and X4 is Med
 and SOP is Med and TERM is Med
 then Char is five;

if X1 is Max and X2 is Med and X3 is Small and X4 is Small
 and TERM is Large
 then Char is six;

if X1 is Small and X2 is Small and X3 is Large and X4 is Small
 and SOP is Small
 then Char is seven;

if X1 is Max and X2 is Med and X3 is Max and X4 is Large
 and SOP is Max and TERM is Max
 then Char is eight;

if X1 is Med and X2 is Small and X3 is Max and X4 is Large
 and SOP is Large
 then Char is nine;

if X1 is Small and X2 is Small and X3 is Med and X4 is Small
 and SOP is Med and TERM is Max
 then Char is SS1;

if X1 is Med and X2 is Med and X3 is Max and X4 is Med
 and SOP is Large and TERM is Max
 then Char is SS2;
```

AN1220/D

```
if X1 is Med and X2 is Med and X3 is Max and X4 is Med
 and SOP is Max and TERM is Max
 then Char is SS3;

if X1 is Small and X2 is Small and X3 is Med and X4 is Small
 and SOP is Small and TERM is Max
 then Char is SS4

end
```

LISTING COMMENTS

The listing is a single text source file composed in the FIDE Inference Language, using the FIDE text editor. Source files must have a DOS extension of .FIL. The FIDE compiler generates a fuzzy inference unit, or FIU, from the source file. Source files specify the fuzzy inference method, define the range of each input variable and all the input membership functions associated with each variable, define the range of each output variable and all output membership functions associated with each variable, and define the fuzzy rules that relate input and output membership functions.

FIDE Inference Language (FIL) is defined in the FIDE Reference Manual. The use of FIL is described in the FIDE User's Manual. As the listing shows, dollar signs ($) precede comments.

The first uncommented line in the listing is:

> fiu tvfi *8

This line specifies that the fuzzy inference unit uses the truth value flow inference (TVFI) method and employs a calculation precision of 8 bits, or 256 divisions. TVFI is the least-complicated of the available calculation methods, and produces the smallest object code image. TVFI produces "singleton" output values that snap crisply from one output state to the next.

The next uncommented line is:

> invar X1 "delta pixels" :2 () 30 [

This line specifies the name of an input variable as X1, measured in units of delta pixels. The valid range of values for X1 is defined as 2 to 30. The set of parentheses specifies that X1 values are represented over the range of 2 to 30 with the full precision of 256 values. An open left-handed square bracket begins the list of input membership function definitions.

The first input membership function definition is:

> Small (@2,1, @11,1, @13,0),

The name of the input membership function is Small. The function is defined by the contents of the parenthetical set. Small starts with the value 1 at the smallest valid value (2). Small remains 1 until 11 delta pixels, when it begins to ramp down, eventually reaching 0 at 13 delta pixels. The comma following the parenthetical set indicates that more input membership functions follow.

The end of the input membership function list is denoted by a closing right-handed square bracket. The end of the input variable definition is denoted by a semicolon. Input variable definitions for X2, X3, X4, SOP, and TERM follow.

Output variable definition begins with the line:

> outvar Char "Character" :0 () 14 * (

The name of the output variable is Char. The variable is assigned the meaningless unit of Character. The range of Char is specified as 0 to 14. The asterisk specifies centroid defuzzification. The open left-handed parenthesis that ends the line starts the list of output membership functions.

The end of the membership function list is denoted by a closing right-handed parenthesis. The end of the output variable definition is denoted by a semicolon. Notice that Char is defined from 0 to 14, while the values assigned to each character recognized ranges from 1 to 14. Thus, if no character is recognized, OCA returns the value 0.

The fuzzy rules that follow the variable definitions are taken directly from Table 7. Each rule is separated from the next by a semicolon. The final rule has no semicolon. It is followed by the FIL statement "end".

The FIDE debugger provides several functions to test for proper operation of the FIU. The analyzer portion of the debugger provides surface and contour maps and cross-sections of input-output relationships, making it easy to visualize how changes in inputs affect outputs. Surface map display mode shows the three-dimensional surface created by the interaction of two input variables with an output. Thus, a surface display cannot be used to show all six input variables simultaneously, but is nonetheless useful in showing system behavior around known areas of concern, such as the overlap area between characters 2 and 5.

The debugger simulator function was also very helpful in testing OCA. The simulator reads a user-defined input data file with a DOS extension of .FDL, generates an output for every combination of inputs specified, and then displays the state of each graphically.

FIDE also provides a system-level simulation capability, called Composer. A system can be composed, or made to incorporate multiple FIU, FOU, and FEU. FIDE operation units (FOU) reformat data passing between multiple FIU, allowing each FIU to represent data independently. FIDE execution units (FEU) simulate the interface to the external world. FEU are written in the C language and typically provide feedback from system outputs or intermediate points to system inputs or intermediate points. Since OCA is not a closed loop system, the composer was not used in testing.

TESTING RESULTS

OCA is a complex fuzzy system with six input variables, one output variable, twenty-three input membership functions, fifteen output membership functions and fourteen fuzzy rules. This complexity makes it difficult to estimate the output for a given set of inputs intuitively.

The most productive testing technique for fuzzy systems is empirical. To test OCA, a text file of test data was supplied to the OCA simulator and generated outputs were compared to the expected outputs.

A group of fourteen data sets was created as a starting point. This group is shown in the listing below. The input data sets correspond, in order, to the fourteen fuzzy rules in the fuzzy inference unit listing. Therefore, proper selection of input values X1 through X4, SOP and TERM should generate the outputs 0 through 9, SS1, SS2, SS3, and SS4, in that order. The values shown are center points for the fuzzy set specified by a given rule, and thus represent a set of input data that OCA should find easy to classify.

This group of fourteen data sets was then duplicated and modified to check for proper operation across the range of character data specified in Tables 3, 5, and 6. The listing shows only the initial fourteen data sets. In each group of data sets, one input variable (say, X1) was taken to the lower limit, with all others held at the upper limit.

Over the entire suite of 112 worst-case simulated data sets, OCA generated two outputs that would have required a second read. These two cases generated outputs that were outside the imposed definition of expected output ±0.13. Importantly, no data sets were incorrectly labelled as being a different character. These worst-case data sets would probably never be encountered in the real world, since transitions, SOP, and TERM would all tend to deviate from mean values in the same direction. Nonetheless, further manipulation of membership functions and rule definitions might make OCA even more robust.

AN1220/D

17

```
;Filename: OCA.FDL
$
$Simulator file for OCA.FIL

X1:2()30,X2:-30()-2,X3:2()30,X4:-30()-2,SOP:60()220,TERM:8()20;


23,-18,18,-23,153,18;
23,-23,00,-00,121,11;
15,-08,08,-15,107,11;
23,-23,00,-00,137,13;
18,-16,08,-11,135,15;
15,-08,08,-16,119,13;
23,-16,05,-08,150,16;
08,-06,12,-09,084,13;
23,-16,16,-23,197,17;
13,-08,19,-23,154,16;
10,-10,10,-10,111,18;
16,-16,16,-16,148,17;
15,-15,16,-16,182,18;
08,-08,08,-08,086,17;
```

GENERATING REAL TIME CODE

FIDE provides a real-time code generator that converts FIDE source code into assembler source code. The code generator prompts for MCU type, then creates an appropriate output file of the same name as the .FIL file, but with an .ASM extension. For OCA, M68HC11 real-time code was generated. The resulting OCA.ASM file was assembled with the IASM11 assembler.

The assembled object code size of OCA is $562 (1123 decimal) bytes. This includes both the fuzzy inference engine and application-specific code. The data preprocessor is not included in this total, but the combined size of preprocessor and fuzzy code should be less than 6 Kbytes.

The code was assembled with an ORG address of $D000, with RAM starting at $0000. X1, X2, X3, and X4 are updated by the preprocessor at locations $0000 to $0003. SOP is updated at $0004 and TERM at $0005. The fuzzy engine output, Char, is updated at location $0006.

This FIU expects fuzzy inputs to be normalized (expressed as a function of input range and step length). The preprocessor must perform this function. Likewise, the FIU generates a normalized output which must be de-normalized by the main program. The normalized input value (n) is defined by:

$$n =$$

The real value of the normalized output is defined by:

$$\text{real value} =$$

SUMMARY

OCA demonstrates the power of fuzzy logic when applied to problems like pattern recognition. The fuzzy logic implementation uses very little memory, making it possible to provide full functionality within the limited confines of MCU ROM space. Since fewer lines of code are executed, microprocessor performance requirements are reduced. In addition, developing a smaller amount of code takes much less time. The graphical user interface, debug facilities and ROM-able inference engine further simplify development.

OCA was developed to provide data as a "black box" function for a host program that would also handle document movement, communication to external computers, and other tasks. In a realized system, OCA would be invoked during optical sensor integration time, after 22 data slices had been obtained. The pre-processor would update the six input variables at location $0000. Very shortly thereafter, the fuzzy engine would update the output at location $0006 with the code corresponding to the character recognized. The host program would take the output character value and copy it into a string of values representing the account number or other data on the document being read.

OCA should operate properly, even with moderate amounts of document skew and misalignment, since these cases were included with the data sets OCA defines. To provide additional guardbanding, the fuzzy sets generally have considerably more latitude than the data sets they incorporate. Therefore, OCA should also be somewhat tolerant of lighting conditions, document background, and printing variations.

REFERENCES

The following publications provide a thorough background in fuzzy logic, from fundamentals to advanced applications.

1. Anderson, Ken, "Control Systems Sample Life In The Fuzzy Lane," *Personal Engineering and Instrumentation News*, October 1992, pg 78.
2. Brubaker, David I., "Fuzzy Logic Basics: Intuitive Rules Replace Complex Math," *EDN*, June 18, 1992, pg 111.
3. Brubaker, David I. and Cedric Sheerer, "Fuzzy Logic System Solves Control Problem," *EDN*, June 18, 1992, pg 121.
4. Kosko, Bart, *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.
5. Schwartz, Daniel G. and George J. Klir, "Fuzzy Logic Flowers In Japan," *IEEE Spectrum*, July 1992, pg 32.
6. Sibigtroth, James M., "Creating Fuzzy Micros," *Embedded Systems Programming*, December 1991.
7. Williams, Tom, "Fuzzy Logic Is Anything But Fuzzy," *Computer Design*, April 1992, pg 113.

AN1220/D

19

Freescale Semiconductor, Inc.

freescale™
semiconductor

For More Information On This Product,
Go to: www.freescale.com