Center For Reliable Computing

TECHNICAL NOTE

Synthesis Tool for Ethernet PAM (Pulse Amplitude Modulator) Encoder

Bharat M. Reddy

CRC TR04-04
Oct 2004

Center for Reliable Computing
Gates Bldg. 2A, Rm #236
Stanford University
Stanford, California 94305-9020

Abstract:

Ethernet Networks have increased in popularity over the years; they have become the most commonly used LAN technology worldwide. More than 85% of all installed network connections are Ethernet, according to the International Data Corporation (IDC, 2000). They have been found in schools, office places, and even homes. Ethernet Networks' speeds have increased over time, and in order to accommodate the growing number of users, faster and higher bandwidth models have been developed. Gigabit Ethernet, which has a data rate of 1 Gbps, has evolved from the original 10Mpbs and 100Mbps Ethernet standards. The Gigabit Ethernet over copper [IEEE 802.3ab] standard adopted a 5-level PAM for encoding. Tools were written in PERL to synthesize the PAM Bit Symbol mapping as specified by the IEEE 802.3ab into Gates. Using the tools, different coding schemes were evaluated and an optimal implementation that was 40% more area efficient compared to the worst-case implementation was reached.

Keywords: *Ethernet, Gigabit Ethernet, PAM encoding, PAM Bit Symbol Mapping,* and *PERL*

Acknowledgements:

I would like to thank Stanford University for giving me the opportunity to work on this project as a summer intern.

Imprimatur: Nirmal R. Saxena

Synthesis Tool for Ethernet PAM (Pulse Amplitude Modulator) Encoder

Bharat M. Reddy

CRC Technical Note No.
October 2004

CENTER FOR RELIABLE COMPUTING
Gates Bldg. 2A, Room #236
Computer System Laboratory
Department of Electrical Engineering
Stanford University
Stanford, California 94305-9020

ABSTRACT

Ethernet Networks have increased in popularity over the years; they have become the most commonly used LAN technology worldwide. More than 85% of all installed network connections are Ethernet, according to the International Data Corporation (IDC, 2000). They have been found in schools, office places, and even homes. Ethernet Networks' speeds have increased over time, and in order to accommodate the growing number of users, faster and higher bandwidth models have been developed. Gigabit Ethernet, which has a data rate of 1 Gbps, has evolved from the original 10Mpbs and 100Mbps Ethernet standards. The Gigabit Ethernet over copper [IEEE 802.3ab] standard adopted a 5-level PAM for encoding. Tools were written in PERL to synthesize the PAM Bit Symbol mapping as specified by the IEEE 802.3ab into Gates. Using the tools, different coding schemes were evaluated and an optimal implementation that was 40% more area efficient compared to the worst-case implementation was reached.

Keywords: *Ethernet, Gigabit Ethernet, PAM encoding, PAM Bit Symbol Mapping,* and *PERL*

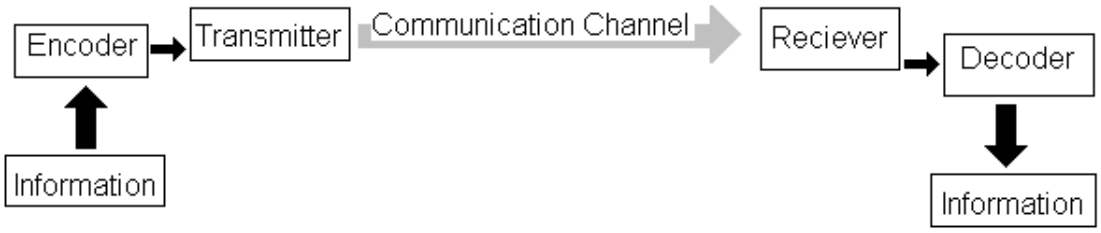
Table of Contents

Table of Contents: Programs

Table of Contents: IEEE 802.3ab PAM Specifications

1.0 Introduction:

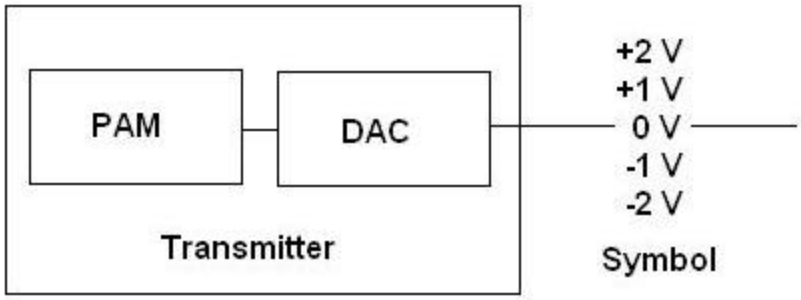
Ethernet Networks have increased in popularity over the years; they have become the most commonly used LAN technology worldwide. More than 85% of all installed network connections are Ethernet according to International Data Corporation (DC, 2000). They have been found in schools, office places, and even homes. Ethernet Networks' speeds have increased over time, and in order to accommodate the growing number of users, faster and higher bandwidth models have been developed. Gigabit Ethernet, which has a data rate of 1 Gbps, has evolved from the original 10Mpbs and 100Mbps Ethernet standards. The Gigabit Ethernet over copper [IEEE 802.3ab] standard adopted a 5-level PAM for encoding.



The figure shows a basic telecommunication model. Information is sent to an encoder that converts the data into binary code, which can be transmitted later. The communication channel can be wires or air (in the case of wireless networking). The information is received and then decoded.

2.0 PAM Encoder:

PAM stands for Pulse Amplitude Modulation. In 5-level PAM, each transmitted symbol represents one of five different levels (-2,-1, 0, +1, +2). The PAM Encoder specification shows Bit-to-Symbol Mapping for odd and even sets. In the figure below, DAC stands for Digital to Analog Converter. It takes the binary coded data from the PAM Encoder and converts it into Symbols that are transmitted.



3.0 Implementation:

The object is to synthesize Bit Symbol Mapping into gates that would later be used in chip development and design. Using the PERL Scripting Language, develop tools to automate the intermediate processes of the gate synthesis procedure. These intermediate steps include merging two ASCII tables, which represent Bit Symbol Mapping, into one table, applying binary coding scheme to the five voltage levels, converting into a Binary I/O table, minimizing the table through the Espresso program and converting the Espresso output into gates. This process can be repeated using different coding schemes. The resulting number of gates for different coding schemes can be compared to find out which is the most efficient scheme. The steps taken to synthesize the PAM encoder are as follows:

Step 1

Write a file using Perl that takes two ASCII files, which contains the PAM specification, and combines them into a single file. These tables were taken from http://www.ieee802.org/3/publication/index.html.

Step 2

Come up with a coding scheme to represent the 5 voltages. Write another tool to convert the 5 output voltages from the joint file into binary code using the coding scheme.

Step 3

Run the output files through Espresso. This is a program that minimizes the coded file, in part by removing redundancies within the file.

Step 4

Write a synthesis tool that reads the espresso product files and coverts them into gates.

Step 5
Use different coding schemes and check which gives the lowest gate count.

4.0 Program Descriptions:

parse2.pl

This program reads from table 40_1 and table 40_2 (PAM specification) and combines the data into a file called joint.txt. It combines it in such a manner so that the 3 bit (i.e. 001) data is printed in front of the six bit (i.e 000000) data. The symbols (i.e. +2,+1,0,-1,-2) are then printed beside it.

coding.pl

Coding.pl reads from the joint.txt file and writes to three separate output files. The program uses a while loop to traverse the joint.txt file. It reads each line and then codes the voltage numbers into 3 separate codes. Each code translates the voltage number into a different 3 bit value; for example, code 1 might translate "-2" into 101, whereas code 2 might translate "-2" into 010. The different codes are written to their respective text file. These 3 files are then run through the program Espresso.

patterncode.pl

After each of the resulting coding schemes (from coding. pl) is passed into the Espresso tool, it is translated into a Product File. This program reads the Product file and whenever it finds a "1" in the output it prints the number of the output (i.e. 00100 - the 1 is output 2) and then searches the input value for 1s and 0s. If a 0 is found it prints ~(not) and the input number; however, if a 1 is found, it simply prints the input number. All of these numbers are linked together by ands or "&"s (i.e 10--1-0- becomes *Input0 & ~Input1 & Input4 & ~Input6.* *note: Spaces and Italics are added for clarity*).

Note: There are multiple product files, one for each coding scheme

fixpatterncode.pl

This program takes the result of patterncode.pl and condenses it. Essentially, patterncode.pl returns a file that has many duplicates due to multiple outputs at the same place. This program finds all the duplicate output numbers (note: duplicate output numbers NOT duplicate outputs) and joins them together by adding an Or in-between them.

order.pl

This orders the condensed result from fixpatterncode.pl by reading all the data into an array and sorting it using a selection sort. The resulting file is the condensed data ordered from lowest output number to greatest.

5.0 Results:

The Voltage levels were coded into 3 different schemes. Each voltage had a corresponding 3 bit binary code. Code 1 used a "1" in the first digit (the 2^2 place) to indicate that the number was negative, while using a "0" in the first digit to indicate that the number was positive. Code 2 did the opposite, as "1" meant positive and "0" meant negative. It also changed the value of zero to 001 (from 000). Code three made the odd binary numbers negative and the even binary numbers positive (i.e. 011 = 3 → -2; 110 = 6 → +2).

Voltages	Code 1	Code 2	Code 3
- 2	110	011	011
-1	101	010	001
0	000	001	000
+1	001	110	010
+2	010	111	110
Total Gates	84	134	90

As seen in the table above, coding scheme 1 gave the best result. It gave almost a 40% reduction from worst case.

6.0 Summary:

A synthesis tool suite was developed using PERL Scripting language. This tool suite automated the process of converting the specification of the PAM encoder into Gates. In addition, using the synthesis tool suite, different coding schemes were explored and an optimal implementation that was 40% more area efficient compared to the worst-case implementation was reached.

1.0 Parse2.pl

```
#!C:\Perl\bin\perl

#This program takes the two tables 40_1 and 40_2 and joins them into one text file

open (writefile, ">joint2.txt") ||
        die ("Unable to open write file\n");
open (readfile, "table40_1.txt") ||
        die ("Unable to open read file\n");
open (readfile2, "table40_2.txt")||
        die ("Unable to open read file\n");

$line = <readfile>;
$line2 = <readfile2>;
```

```
while ($line ne "")
{
	if(length($line) == 76)
	{
		$sixbits = substr ($line, 13, 6);

		$mark001 = "001 ";
		$mark011 = "011 ";
		$mark101 = "101 ";
		$mark111 = "111 ";

		$mark000 = "000 ";
		$mark010 = "010 ";
		$mark100 = "100 ";
		$mark110 = "110 ";

		$biz000 = substr ($line, 23, 11);
		$biz010 = substr ($line, 37, 10);
		$biz100 = substr ($line, 51, 10);
		$biz110 = substr ($line, 65, 10);
	}
	else
	{
		$line = <readfile>;
	}

	if ($line2 ne "")
	{
		if(length($line2)==65)
		{
			$biz001 = substr ($line, 23, 11);
			$biz011 = substr ($line, 37, 10);
			$biz101 = substr ($line, 51, 10);
			$biz111 = substr ($line, 65, 10);

			print writefile "$mark000 $sixbits $biz000 \n";

			print writefile "$mark001 $sixbits $biz001 \n";

			print writefile "$mark010 $sixbits $biz010 \n";

			print writefile "$mark011 $sixbits $biz011 \n";

			print writefile "$mark100 $sixbits $biz100 \n";

			print writefile "$mark101 $sixbits $biz101 \n";
```

```
                print writefile "$mark110 $sixbits $biz110 \n";

                print writefile "$mark111 $sixbits $biz111 \n\n";
        }
        else
        {
                $line2 = <readfile2>;
        }

    }
    else
    {

        print writefile "$mark000 $sixbits $biz000 \n";
        print writefile "$mark010 $sixbits $biz010 \n";
        print writefile "$mark100 $sixbits $biz100 \n";
        print writefile "$mark110 $sixbits $biz110 \n";

    }

    $line = <readfile>;
    $line2 = <readfile2>;
}
```

2.0 Coding.pl

```
#!C:\Perl\bin\perl

open (writefile, ">coding.txt") ||
    die ("Unable to open write file\n");
open (writefile2, ">coding2.txt") ||
    die ("Unable to open write file\n");
open (writefile3, ">coding3.txt") ||
    die ("Unable to open write file\n");
open (writefile4, ">coding4.txt") ||
    die ("Unable to open write file\n");
open (readfile, "joint.txt") ||
    die ("Unable to open read file\n");


$line = <readfile>;
while ($line ne "")
{

    $ninebits = substr($line, 0, 9);    #this substring is the 9 bit input value
```

```
	$threebits = substr($line, 0, 3);		#This substring is the 3 bit value at the beginning of each input value
	$xbits = substr($line, 3, 6);		#This substring is the remaining 6 bit value at the end of the input value

	# Checks to see if there are any "X"s in the input
	# IF there are it replaces them with dashes because the Espresso tool through which
	# the outputs will be run through can only read dashes and not Xs
	if($xbits == "XXXXXX")
	{
		if($xbits ne "000000"){		print writefile "$threebits------ ";}
		else{print writefile "$ninebits ";}
	}
	else
	{
		print writefile "$ninebits ";
	}

	$data = substr($line, 9, 11); # The output value which is the frequency numbers

	# This is the first code, each frequency number is given a corresponding 3 bit value
	# -2 = 110
	# -1 = 101
	#  0 = 000
	# +1 = 001
	# +2 = 010


	for($x=0; $x<11; $x=$x+3)
	{

		$test = substr($data, $x, 2);		# The test substring is the numbers in output value.
		$dashtest = substr($data, 5, 1); #checks if there is a dash instead of a number
		#print writefile " ($test) ";
		#print writefile " ($dashtest) ";

		if($test ne "")
		{
			if($test == "-2")
			{
				print writefile "110";
			}
```

```
                        elsif($test == "-1")
                        {
                                print writefile "101";
                        }
                        # If there was a dash then 3 dashes are written to the file, otherwise
                        # the normal code for the number zero is written. This dash test
could
                        # have been implemented with any other number, but i chose to
check it with
                        # zero.
                        elsif($test == "0")
                        {
                                if($dashtest ne "-") {print writefile "000";}
                                else{print writefile "---";}
                        }
                        elsif($test == "+1")
                        {
                                print writefile "001";
                        }
                        elsif($test == "+2")
                        {
                                print writefile "010";
                        }
                        elsif($test == "")
                        {
                                print writefile "---";
                        }
                }
        }

        # the coded line is the printed to the write file
        print writefile " \n";

        #This process is repeated for all of the other coding schemes

# code 2

        $ninebits = substr($line, 0, 9);
        $threebits = substr($line, 0, 3);
        $xbits = substr($line, 3, 6);

        if($xbits == "XXXXXX")
        {
                if($xbits ne "000000"){
                print writefile2 "$threebits------ ";}
                else{print writefile2 "$ninebits ";}
```

```
}
else
{
	print writefile2 "$ninebits ";
}

#print writefile2 "$data \n\n";
$data = substr($line, 9, 11);

#  CODE 2
# -2 = 011
# -1 = 010
#  0 = 001
# +1 = 110
# +2 = 111

for($x=0; $x<11; $x=$x+3)
{
	$test = substr($data, $x, 2);
	$dashtest = substr($data, 5, 1);
	#print writefile2 " ($test) ";
	#print writefile2 " ($dashtest) ";

	if($test ne "")
	{
		if($test == "-2")
		{
			print writefile2 "011";
		}
		elsif($test == "-1")
		{
			print writefile2 "010";
		}
		elsif($test == "0")
		{
			if($dashtest ne "-") {print writefile2 "001";}
			else{print writefile2 "---";}
		}
		elsif($test == "+1")
		{
			print writefile2 "110";
		}
		elsif($test == "+2")
		{
			print writefile2 "111";
		}
```

```
        }
}
print writefile2 " \n";


$ninebits = substr($line, 0, 9);
$threebits = substr($line, 0, 3);
$xbits = substr($line, 3, 6);

if($xbits == "XXXXXX")
{
        if($xbits ne "000000"){         print writefile3 "$threebits------ ";}
        else{print writefile3 "$ninebits ";}
}
else
{
        print writefile3 "$ninebits ";
}

#print writefile3 "$data \n\n";
$data = substr($line, 9, 11);

#  CODE 3
# -2 = 011
# -1 = 001
#  0 = 000
# +1 = 010
# +2 = 110

for($x=0; $x<11; $x=$x+3)
{
        $test = substr($data, $x, 2);
        $dashtest = substr($data, 5, 1);
        #print writefile4 " ($test) ";
        #print writefile4 " ($dashtest) ";

        if($test ne "")
        {
                if($test == "-2")
                {
                        print writefile4 "011";
                }
                elsif($test == "-1")
                {
                        print writefile4 "001";
                }
```

```
                        elsif($test == "0")
                        {
                                if($dashtest ne "-") {print writefile4 "000";}
                                else{print writefile4 "---";}
                        }
                        elsif($test == "+1")
                        {
                                print writefile4 "010";
                        }
                        elsif($test == "+2")
                        {
                                print writefile4 "110";
                        }
                }
        }
        print writefile4 " \n";
        $line = <readfile>;
}
```

3.0 Patterncode.pl

```
#!C:\Perl\bin\perl

#Note: this is just one of four pattercode files. Each one is identical except for
#       the coding file which is being read.
open (readfile, "codingresult.txt") ||
        die ("Unable to open write file\n");
open (writefile, ">printandor.txt") ||
        die ("Unable to open printandor file\n");

$line = <readfile>;

# Skips the first 3 lines of the file because they contain values
# that represent the number of inputs, outputs and total products
# in the file.
for($skip=0; $skip<3; $skip++)
{
        $line = <readfile>;
}

while($line ne "")
{
        $output = substr($line,10,12); #The output string
        $input = substr($line, 0, 9);  #the input string

        # This for loop checks the output string for a 1
```

```
	for($x=0; $x<length($output); $x++)
	{
		$subby = substr($output, $x,1);
		# IF there is a 1 then it goes into the input string
		if($subby == "1")
		{
			print writefile "o$x = "; #prints the output number
			$counter = 0;
			for($y=0; $y<length($input); $y++)
			{

				$test = substr($input, $y,1); #traverses one character at a
time
				if($test != "-" || $test eq "0")
				{
					if($test == "0")
					{
						if($counter > 0){print writefile "&";} #prints
&(and) between terms
						$counter++;
						print writefile "~s$y"; #if the char is 0 then
it prints ~(not) #in front of the term. The "s" means  #input
					}
					else
					{
						if($counter > 0){print writefile "&";} #prints
&(and) between terms
						$counter++;
						print writefile "s$y";
					}
				}
			}
			print writefile "\n"; #prints a space
		}
	}
	$line = <readfile>; #Reads the next line in
}
```

4.0 Fixpatterncode2.pl

```
#!C:\Perl\bin\perl
open (readfile, "printandor.txt") ||
	die ("Unable to open printandor file\n");
open (readfile2, "printandor2.txt") ||
	die ("Unable to pen printandor2 file\n");
open (writefile, ">finalandor.txt") ||
```

```
        die ("Unable to open printandor file\n");

$line = <readfile>;
$line2 = <readfile2>;
@string1arr = ();  #Array that keeps track of duplicate outputs
$contains = false; #This checks if an output is already contained in the array
$length=@string1arr; #Length of the array

while($line2 ne "")
{
        $string1 = "";
        $string2 = "";
        $counter = 0;

        for($x=0; $x<length($line2)-1; $x++)
        {
                #Sets everything before the = sign to string one
                $char = substr($line2, $x, 1);
                #Sets everything after the = sign to string 2
                if($char eq '=')
                {
                        $counter = 5; #changes counter-shows this is after the equals sign
                        $char = substr($line2, $x+1, 1);
                }
                if($counter == 0){$string1 = $string1.$char;} #char added to string 1
                if($counter == 5){$string2 = $string2.$char;} #chat added to string 2
        }

        for($x=0;$x<$length;$x++)
        {
                if($string1arr[$x] eq $string1)
                {
                        $contains = true;
                }
        }

        if($contains eq true)
        {
                $line2 = <readfile2>;
                $line = <readfile>;
                $contains = false;
        }
        else
        {
                push(@string1arr, $string1);
                $length++;
```

```
		$line = <readfile>;
		$counter3 = 0;
		print writefile "$string1: ";

		while($line ne "")
		{
			$string1a = "";
			$string2a = "";
			$counter2 = 0;

			for($x=0; $x<length($line)-1; $x++)
			{
			$char1 = substr($line, $x, 1);

				if($char1 eq '=')
				{
					$counter2 = 4;
					$char1 = substr($line, $x+1, 1);
				}
				if($counter2 == 0){$string1a = $string1a.$char1;}
				if($counter2 == 4){$string2a = $string2a.$char1;}
			}

			if($string1 eq $string1a)
			{
				if($counter3==0){print writefile $string2;}
				$counter3=1;
				print writefile " or$string2a";
			}
			$line = <readfile>;
		}

		print writefile "\n";
		close(readfile);
		open (readfile, "printandor.txt") ||
			die ("Unable to open printandor file\n");

		$line2 = <readfile2>;

	}
}
```

5.0 Order.pl

```
#!C:\Perl\bin\perl
```

```
open (readfile, "finalandor2.txt") ||
        die ("Unable to open finalandor file\n");
open (writefile, ">inorderfao2.txt") ||
        die ("Unable to open inorderfao file\n");

$line = <readfile>;

# This program takes the condensed file and correctly orders numerically.

@data = ();
@inorder = ();
$dataamount = 0;

while($line ne "")
{
        push(@data, substr($line,1,2));
        push(@inorder, $line);
        $dataamount++;
        $line = <readfile>;
}

for($i = 0; $i < $dataamount -1; $i++)
{
        $minIndex = $i;
        for($k = $i +1; $k <$dataamount; $k++)
        {
                if(substr($inorder[$k],1,2) < substr($inorder[$minIndex],1,2))
                {
                        $minIndex = $k;
                }
        }
        $temp = $inorder[$i];
        $inorder[$i] = $inorder[$minIndex];
        $inorder[$minIndex] = $temp;
}

print writefile @inorder;
```

1.0 Table 40-1

Table 40-1 Bit-to-Symbol Mapping (Even Subsets)
==========

Sdn[6:8] = [000]	Sdn[6:8] = [010]	Sdn[6:8] = [100]	Sdn[6:8] = [110]

Condition	Sdn[5:0]	TAn,TBn, TCn,TDn	TAn,TBn, TCn,TDn	TAn,TBn, TCn,TDn	TAn,TBn, TCn,TDn
Normal	000000	0, 0, 0, 0	0, 0,+1,+1	0,+1,+1, 0	0,+1, 0,+1
Normal	000001	-2, 0, 0, 0	-2, 0,+1,+1	-2,+1,+1, 0	-2,+1, 0,+1
Normal	000010	0,-2, 0, 0	0,-2,+1,+1	0,-1,+1, 0	0,-1, 0,+1
Normal	000011	-2,-2, 0, 0	-2,-2,+1,+1	-2,-1,+1, 0	-2,-1, 0,+1
Normal	000100	0, 0,-2, 0	0, 0,-1,+1	0,+1,-1, 0	0,+1,-2,+1
Normal	000101	-2, 0,-2, 0	-2, 0,-1,+1	-2,+1,-1, 0	-2,+1,-2,+1
Normal	000110	0,-2,-2, 0	0,-2,-1,+1	0,-1,-1, 0	0,-1,-2,+1
Normal	000111	-2,-2,-2, 0	-2,-2,-1,+1	-2,-1,-1, 0	-2,-1,-2,+1
Normal	001000	0, 0, 0,-2	0, 0,+1,-1	0,+1,+1,-2	0,+1, 0,-1
Normal	001001	-2, 0, 0,-2	-2, 0,+1,-1	-2,+1,+1,-2	-2,+1, 0,-1
Normal	001010	0,-2, 0,-2	0,-2,+1,-1	0,-1,+1,-2	0,-1, 0,-1
Normal	001011	-2,-2, 0,-2	-2,-2,+1,-1	-2,-1,+1,-2	-2,-1, 0,-1
Normal	001100	0, 0,-2,-2	0, 0,-1,-1	0,+1,-1,-2	0,+1,-2,-1
Normal	001101	-2, 0,-2,-2	-2, 0,-1,-1	-2,+1,-1,-2	-2,+1,-2,-1
Normal	001110	0,-2,-2,-2	0,-2,-1,-1	0,-1,-1,-2	0,-1,-2,-1
Normal	001111	-2,-2,-2,-2	-2,-2,-1,-1	-2,-1,-1,-2	-2,-1,-2,-1
Normal	010000	+1,+1,+1,+1	+1,+1, 0, 0	+1, 0, 0,+1	+1, 0,+1, 0
Normal	010001	-1,+1,+1,+1	-1,+1, 0, 0	-1, 0, 0,+1	-1, 0,+1, 0
Normal	010010	+1,-1,+1,+1	+1,-1, 0, 0	+1,-2, 0,+1	+1,-2,+1, 0
Normal	010011	-1,-1,+1,+1	-1,-1, 0, 0	-1,-2, 0,+1	-1,-2,+1, 0
Normal	010100	+1,+1,-1,+1	+1,+1,-2, 0	+1, 0,-2,+1	+1, 0,-1, 0
Normal	010101	-1,+1,-1,+1	-1,+1,-2, 0	-1, 0,-2,+1	-1, 0,-1, 0
Normal	010110	+1,-1,-1,+1	+1,-1,-2, 0	+1,-2,-2,+1	+1,-2,-1, 0
Normal	010111	-1,-1,-1,+1	-1,-1,-2, 0	-1,-2,-2,+1	-1,-2,-1, 0
Normal	011000	+1,+1,+1,-1	+1,+1, 0,-2	+1, 0, 0,-1	+1, 0,+1,-2
Normal	011001	-1,+1,+1,-1	-1,+1, 0,-2	-1, 0, 0,-1	-1, 0,+1,-2
Normal	011010	+1,-1,+1,-1	+1,-1, 0,-2	+1,-2, 0,-1	+1,-2,+1,-2
Normal	011011	-1,-1,+1,-1	-1,-1, 0,-2	-1,-2, 0,-1	-1,-2,+1,-2
Normal	011100	+1,+1,-1,-1	+1,+1,-2,-2	+1, 0,-2,-1	+1, 0,-1,-2
Normal	011101	-1,+1,-1,-1	-1,+1,-2,-2	-1, 0,-2,-1	-1, 0,-1,-2
Normal	011110	+1,-1,-1,-1	+1,-1,-2,-2	+1,-2,-2,-1	+1,-2,-1,-2
Normal	011111	-1,-1,-1,-1	-1,-1,-2,-2	-1,-2,-2,-1	-1,-2,-1,-2
Normal	100000	+2, 0, 0, 0	+2, 0,+1,+1	+2,+1,+1, 0	+2,+1, 0,+1
Normal	100001	+2,-2, 0, 0	+2,-2,+1,+1	+2,-1,+1, 0	+2,-1, 0,+1
Normal	100010	+2, 0,-2, 0	+2, 0,-1,+1	+2,+1,-1, 0	+2,+1,-2,+1
Normal	100011	+2,-2,-2, 0	+2,-2,-1,+1	+2,-1,-1, 0	+2,-1,-2,+1
Normal	100100	+2, 0, 0,-2	+2, 0,+1,-1	+2,+1,+1,-2	+2,+1, 0,-1
Normal	100101	+2,-2, 0,-2	+2,-2,+1,-1	+2,-1,+1,-2	+2,-1, 0,-1
Normal	100110	+2, 0,-2,-2	+2, 0,-1,-1	+2,+1,-1,-2	+2,+1,-2,-1
Normal	100111	+2,-2,-2,-2	+2,-2,-1,-1	+2,-1,-1,-2	+2,-1,-2,-1
Normal	101000	0, 0,+2, 0	+1,+1,+2, 0	+1, 0,+2,+1	0,+1,+2,+1
Normal	101001	-2, 0,+2, 0	-1,+1,+2, 0	-1, 0,+2,+1	-2,+1,+2,+1
Normal	101010	0,-2,+2, 0	+1,-1,+2, 0	+1,-2,+2,+1	0,-1,+2,+1
Normal	101011	-2,-2,+2, 0	-1,-1,+2, 0	-1,-2,+2,+1	-2,-1,+2,+1

Normal	101100	0, 0,+2,-2	+1,+1,+2,-2	+1, 0,+2,-1	0,+1,+2,-1
Normal	101101	-2, 0,+2,-2	-1,+1,+2,-2	-1, 0,+2,-1	-2,+1,+2,-1
Normal	101110	0,-2,+2,-2	+1,-1,+2,-2	+1,-2,+2,-1	0,-1,+2,-1
Normal	101111	-2,-2,+2,-2	-1,-1,+2,-2	-1,-2,+2,-1	-2,-1,+2,-1
Normal	110000	0,+2, 0, 0	0,+2,+1,+1	+1,+2, 0,+1	+1,+2,+1, 0
Normal	110001	-2,+2, 0, 0	-2,+2,+1,+1	-1,+2, 0,+1	-1,+2,+1, 0
Normal	110010	0,+2,-2, 0	0,+2,-1,+1	+1,+2,-2,+1	+1,+2,-1, 0
Normal	110011	-2,+2,-2, 0	-2,+2,-1,+1	-1,+2,-2,+1	-1,+2,-1, 0
Normal	110100	0,+2, 0,-2	0,+2,+1,-1	+1,+2, 0,-1	+1,+2,+1,-2
Normal	110101	-2,+2, 0,-2	-2,+2,+1,-1	-1,+2, 0,-1	-1,+2,+1,-2
Normal	110110	0,+2,-2,-2	0,+2,-1,-1	+1,+2,-2,-1	+1,+2,-1,-2
Normal	110111	-2,+2,-2,-2	-2,+2,-1,-1	-1,+2,-2,-1	-1,+2,-1,-2
Normal	111000	0, 0, 0,+2	+1,+1, 0,+2	0,+1,+1,+2	+1, 0,+1,+2
Normal	111001	-2, 0, 0,+2	-1,+1, 0,+2	-2,+1,+1,+2	-1, 0,+1,+2
Normal	111010	0,-2, 0,+2	+1,-1, 0,+2	0,-1,+1,+2	+1,-2,+1,+2
Normal	111011	-2,-2, 0,+2	-1,-1, 0,+2	-2,-1,+1,+2	-1,-2,+1,+2
Normal	111100	0, 0,-2,+2	+1,+1,-2,+2	0,+1,-1,+2	+1, 0,-1,+2
Normal	111101	-2, 0,-2,+2	-1,+1,-2,+2	-2,+1,-1,+2	-1, 0,-1,+2
Normal	111110	0,-2,-2,+2	+1,-1,-2,+2	0,-1,-1,+2	+1,-2,-1,+2
Normal	111111	-2,-2,-2,+2	-1,-1,-2,+2	-2,-1,-1,+2	-1,-2,-1,+2

2.0 Table 40-2

Table 40-2 Bit-to-Symbol Mapping (Odd Subsets)
==========

Condition	Sdn[5:0]	Sdn[6:8] = [001] TAn,TBn, TCn,TDn	Sdn[6:8] = [011] TAn,TBn, TCn,TDn	Sdn[6:8] = [101] TAn,TBn, TCn,TDn	Sdn[6:8] = [111] TAn,TBn, TCn,TDn
Normal	000000	0, 0, 0,+1	0, 0,+1, 0	0,+1,+1,+1	0,+1, 0, 0
Normal	000001	-2, 0, 0,+1	-2, 0,+1, 0	-2,+1,+1,+1	-2,+1, 0, 0
Normal	000010	0,-2, 0,+1	0,-2,+1, 0	0,-1,+1,+1	0,-1, 0, 0
Normal	000011	-2,-2, 0,+1	-2,-2,+1, 0	-2,-1,+1,+1	-2,-1, 0, 0
Normal	000100	0, 0,-2,+1	0, 0,-1, 0	0,+1,-1,+1	0,+1,-2, 0
Normal	000101	-2, 0,-2,+1	-2, 0,-1, 0	-2,+1,-1,+1	-2,+1,-2, 0
Normal	000110	0,-2,-2,+1	0,-2,-1, 0	0,-1,-1,+1	0,-1,-2, 0
Normal	000111	-2,-2,-2,+1	-2,-2,-1, 0	-2,-1,-1,+1	-2,-1,-2, 0
Normal	001000	0, 0, 0,-1	0, 0,+1,-2	0,+1,+1,-1	0,+1, 0,-2
Normal	001001	-2, 0, 0,-1	-2, 0,+1,-2	-2,+1,+1,-1	-2,+1, 0,-2
Normal	001010	0,-2, 0,-1	0,-2,+1,-2	0,-1,+1,-1	0,-1, 0,-2
Normal	001011	-2,-2, 0,-1	-2,-2,+1,-2	-2,-1,+1,-1	-2,-1, 0,-2
Normal	001100	0, 0,-2,-1	0, 0,-1,-2	0,+1,-1,-1	0,+1,-2,-2
Normal	001101	-2, 0,-2,-1	-2, 0,-1,-2	-2,+1,-1,-1	-2,+1,-2,-2
Normal	001110	0,-2,-2,-1	0,-2,-1,-2	0,-1,-1,-1	0,-1,-2,-2

Normal	001111	-2,-2,-2,-1	-2,-2,-1,-2	-2,-1,-1,-1	-2,-1,-2,-2
Normal	010000	+1,+1,+1, 0	+1,+1, 0,+1	+1, 0, 0, 0	+1, 0,+1,+1
Normal	010001	-1,+1,+1, 0	-1,+1, 0,+1	-1, 0, 0, 0	-1, 0,+1,+1
Normal	010010	+1,-1,+1, 0	+1,-1, 0,+1	+1,-2, 0, 0	+1,-2,+1,+1
Normal	010011	-1,-1,+1, 0	-1,-1, 0,+1	-1,-2, 0, 0	-1,-2,+1,+1
Normal	010100	+1,+1,-1, 0	+1,+1,-2,+1	+1, 0,-2, 0	+1, 0,-1,+1
Normal	010101	-1,+1,-1, 0	-1,+1,-2,+1	-1, 0,-2, 0	-1, 0,-1,+1
Normal	010110	+1,-1,-1, 0	+1,-1,-2,+1	+1,-2,-2, 0	+1,-2,-1,+1
Normal	010111	-1,-1,-1, 0	-1,-1,-2,+1	-1,-2,-2, 0	-1,-2,-1,+1
Normal	011000	+1,+1,+1,-2	+1,+1, 0,-1	+1, 0, 0,-2	+1, 0,+1,-1
Normal	011001	-1,+1,+1,-2	-1,+1, 0,-1	-1, 0, 0,-2	-1, 0,+1,-1
Normal	011010	+1,-1,+1,-2	+1,-1, 0,-1	+1,-2, 0,-2	+1,-2,+1,-1
Normal	011011	-1,-1,+1,-2	-1,-1, 0,-1	-1,-2, 0,-2	-1,-2,+1,-1
Normal	011100	+1,+1,-1,-2	+1,+1,-2,-1	+1, 0,-2,-2	+1, 0,-1,-1
Normal	011101	-1,+1,-1,-2	-1,+1,-2,-1	-1, 0,-2,-2	-1, 0,-1,-1
Normal	011110	+1,-1,-1,-2	+1,-1,-2,-1	+1,-2,-2,-2	+1,-2,-1,-1
Normal	011111	-1,-1,-1,-2	-1,-1,-2,-1	-1,-2,-2,-2	-1,-2,-1,-1
Normal	100000	+2, 0, 0,+1	+2, 0,+1, 0	+2,+1,+1,+1	+2,+1, 0, 0
Normal	100001	+2,-2, 0,+1	+2,-2,+1, 0	+2,-1,+1,+1	+2,-1, 0, 0
Normal	100010	+2, 0,-2,+1	+2, 0,-1, 0	+2,+1,-1,+1	+2,+1,-2, 0
Normal	100011	+2,-2,-2,+1	+2,-2,-1, 0	+2,-1,-1,+1	+2,-1,-2, 0
Normal	100100	+2, 0, 0,-1	+2, 0,+1,-2	+2,+1,+1,-1	+2,+1, 0,-2
Normal	100101	+2,-2, 0,-1	+2,-2,+1,-2	+2,-1,+1,-1	+2,-1, 0,-2
Normal	100110	+2, 0,-2,-1	+2, 0,-1,-2	+2,+1,-1,-1	+2,+1,-2,-2
Normal	100111	+2,-2,-2,-1	+2,-2,-1,-2	+2,-1,-1,-1	+2,-1,-2,-2
Normal	101000	0, 0,+2,+1	+1,+1,+2,+1	+1, 0,+2, 0	0,+1,+2, 0
Normal	101001	-2, 0,+2,+1	-1,+1,+2,+1	-1, 0,+2, 0	-2,+1,+2, 0
Normal	101010	0,-2,+2,+1	+1,-1,+2,+1	+1,-2,+2, 0	0,-1,+2, 0
Normal	101011	-2,-2,+2,+1	-1,-1,+2,+1	-1,-2,+2, 0	-2,-1,+2, 0
Normal	101100	0, 0,+2,-1	+1,+1,+2,-1	+1, 0,+2,-2	0,+1,+2,-2
Normal	101101	-2, 0,+2,-1	-1,+1,+2,-1	-1, 0,+2,-2	-2,+1,+2,-2
Normal	101110	0,-2,+2,-1	+1,-1,+2,-1	+1,-2,+2,-2	0,-1,+2,-2
Normal	101111	-2,-2,+2,-1	-1,-1,+2,-1	-1,-2,+2,-2	-2,-1,+2,-2
Normal	110000	0,+2, 0,+1	0,+2,+1, 0	+1,+2, 0, 0	+1,+2,+1,+1
Normal	110001	-2,+2, 0,+1	-2,+2,+1, 0	-1,+2, 0, 0	-1,+2,+1,+1
Normal	110010	0,+2,-2,+1	0,+2,-1, 0	+1,+2,-2, 0	+1,+2,-1,+1
Normal	110011	-2,+2,-2,+1	-2,+2,-1, 0	-1,+2,-2, 0	-1,+2,-1,+1
Normal	110100	0,+2, 0,-1	0,+2,+1,-2	+1,+2, 0,-2	+1,+2,+1,-1
Normal	110101	-2,+2, 0,-1	-2,+2,+1,-2	-1,+2, 0,-2	-1,+2,+1,-1
Normal	110110	0,+2,-2,-1	0,+2,-1,-2	+1,+2,-2,-2	+1,+2,-1,-1
Normal	110111	-2,+2,-2,-1	-2,+2,-1,-2	-1,+2,-2,-2	-1,+2,-1,-1
Normal	111000	+1,+1,+1,+2	0, 0,+1,+2	+1, 0, 0,+2	0,+1, 0,+2
Normal	111001	-1,+1,+1,+2	-2, 0,+1,+2	-1, 0, 0,+2	-2,+1, 0,+2
Normal	111010	+1,-1,+1,+2	0,-2,+1,+2	+1,-2, 0,+2	0,-1, 0,+2
Normal	111011	-1,-1,+1,+2	-2,-2,+1,+2	-1,-2, 0,+2	-2,-1, 0,+2
Normal	111100	+1,+1,-1,+2	0, 0,-1,+2	+1, 0,-2,+2	0,+1,-2,+2

Normal	111101	-1,+1,-1,+2	-2, 0,-1,+2	-1, 0,-2,+2	-2,+1,-2,+2
Normal	111110	+1,-1,-1,+2	0,-2,-1,+2	+1,-2,-2,+2	0,-1,-2,+2
Normal	111111	-1,-1,-1,+2	-2,-2,-1,+2	-1,-2,-2,+2	-2,-1,-2,+2