

Implementing V.32bis Viterbi Decoding on the TMS320C62xx DSP

APPLICATION REPORT: SPRA444

*Henry Yiu
Customer Applications Center
Texas Instruments Hong Kong Ltd.*

*Digital Signal Processing Solutions
April 1998*



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Abstract	7
Product Support.....	8
Related Documentation.....	8
World Wide Web	8
Email.....	8
Introduction.....	9
Step 1. Opening the Function	10
Step 2. Euclidean Distance Calculation	11
Step 3. Find Shortest Distances	17
Step 4. Calculate Accumulate Distances	19
Step 5. Trace Backward for Path-State	21
Step 6. Differentiate	23
Step 7. Closing the Function	24
Building and Testing the Code	25
Conclusion	27
Appendix A. V.32BIS Viterbi C Code Implementation	28
Appendix B. V.32Bis Viterbi C62XX Assembly Implementation	31
Appendix C. Main Program to Test the Algorithm.....	45

Figures

Figure 1.	Dividing the Signal Space Diagram into Regions for 14.4kbps – with Path-Index (Y0, Y1, and Y2) = 000	12
Figure 2.	Dividing the Signal Space Diagram into Regions for 12 kbps – with Path-Index (Y0, Y1, and Y2) = 000	13
Figure 3.	Dividing the Signal Space Diagram into Regions for 9.6 kbps – with Path-Index (Y0, Y1, and Y2) = 000	13
Figure 4.	Dividing the Signal Space Diagram into Regions for 7.2 kbps – with Path-Index (Y0, Y1, and Y2) = 000	14
Figure 5.	Structure of the Signal Space Constellation Diagram	15
Figure 6.	Using Registers to Store the TMBack Look-up Table	18
Figure 7.	Calculating the Accumulated Distances	19
Figure 8.	DelayPath Array Structure.....	21
Figure 9.	Differentiation.....	23

Tables

Table 1.	Path-States of V.32bis	11
Table 2.	Byte Size Requirement for Signal Space Constellation Look-up Table.....	16
Table 3.	Clock Cycle Requirements for V.32bis Viterbi Decoding Algorithm.	26

Implementing V.32bis Viterbi Decoding on the TMS320C62xx DSP

Abstract

This paper describes the implementation of the V.32bis decoding algorithm on the Texas Instruments (TI™) TMS320C62xx digital signal processor (DSP). The V.32bis Viterbi decoder algorithm is based on a soft-decision maximum-likelihood decoding technique. (Details on the theory behind this algorithm are described in the TI publication, *DSP Solutions for Telephony and Data/Facsimile Modems*, literature number SPRA073.)

This V.32bis decoding algorithm is written using hand-coded assembly and is C callable. Implementation is divided into seven steps as follows:

- ❑ Step 1. Opening the Function
- ❑ Step 2. Euclidean Distance Calculation
- ❑ Step 3. Find Shortest Distances
- ❑ Step 4. Calculate Accumulate Distances
- ❑ Step 5. Trace Backward for Path-State
- ❑ Step 6. Differentiate
- ❑ Step 7. Closing the Function

Appendix A contains the C code implementation of the V.32bis Viterbi algorithm. Appendix B contains the C62xx assembly code implementation. Appendix C contains the main C program to test the performance of the Viterbi code.

Product Support

Related Documentation

The following list specifies product names, part numbers, and literature numbers of corresponding TI documentation.

- ❑ *TMS320C62x/C67x Programmer's Guide*, February 1998, Literature number SPRU198B
- ❑ Massey, Tim and Lyer, Ramesh, *DSP Solutions for Telephony and Data/Facsimile Modems*, 1997, Literature number SPRA073

World Wide Web

Our World Wide Web site at **www.ti.com** contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to **dsph@ti.com**. Questions receive prompt attention and are usually answered within one business day.



Introduction

In the description of the algorithm that follows, the portion of the C code and assembly code that corresponds to the particular step is referenced with line numbers of the code listing in Appendix A and Appendix B.

The description of the following program is for the assembly code only. The C code can be easily understood by reading the comments listed with the C listing.

Some C62xx assembly instructions require several cycles of delay slots. To fully utilize C62xx performance, the assembly code must be written in pipeline fashion. To document the pipeline code, each assembly instruction is labeled with asterisks at the comment field. The number of asterisks represents the step number to which each instruction belongs.



Step 1. Opening the Function

C Code: Line 51 to 59
Assembly: Line 362 to 385

This step initializes the function for subsequent operation. The normal C calling convention must be followed. Registers must be saved to the stack before being modified.



Step 2. Euclidean Distance Calculation

C Code: Line 61 to 105
 Assembly: Line 387 to 521

The V.32bis Viterbi cost function is the Euclidean distance between the received symbol as mapped on the constellation chart and a predefined point on the same chart. For each symbol received, eight distances to the nearest eight path-states should be generated as the cost function.

The V.32bis specifies a signal space diagram for 14.4, 12, 9.6, 7.2, and 4.8 kbps. A table must be created to store these symbol locations. The scheme described in this report intends to minimize the memory size of these tables and maximize the speed of the cost function calculation.

Under the V.32bis specification, each path-state in the signal space diagram is represented as shown in Table 1:

Table 1. Path-States of V.32bis

Bit Rate	Path-State
14.4 kbps	Y0,Y1,Y2,Q3,Q4,Q5,Q6
12.0 kbps	Y0,Y1,Y2,Q3,Q4,Q5,
9.6 kbps	Y0,Y1,Y2,Q3,Q4
7.2 kbps	Y0,Y1,Y2,Q3
4.8 kbps	Y1n,Y2n

For each point (Y0, Y1, and Y2) from 000 to 111, the shortest distance between the received symbol and the path states must be found. Therefore, there are a total of eight distances. These are Distance[0] to Distance[7], and eight corresponding path-states, State[0] to State[7], to be generated from this step.

The (Y0, Y1, and Y2) portion of the path-state is called the path-index. To avoid confusion with the path-index, the path-state is the path-index plus the bits Q3, Q4, etc.

This scheme divides the signal space diagram into square block regions. First, the region containing the received symbol is determined. Next, the shortest distances between the received symbol and the two closest path-states are calculated and selected.

Figure 1 through Figure 4 show how the signal space diagrams for 14.4 kbps, 12 kbps, 9.6 kbps, and 7.2 kbps are divided. These figures are for path-index = 000. Seven more figures for each bit rate are needed to show how the signal space diagrams are divided for all possible path-indexes.

Figure 1. Dividing the Signal Space Diagram into Regions for 14.4kbps – with Path-Index (Y0, Y1, and Y2) = 000

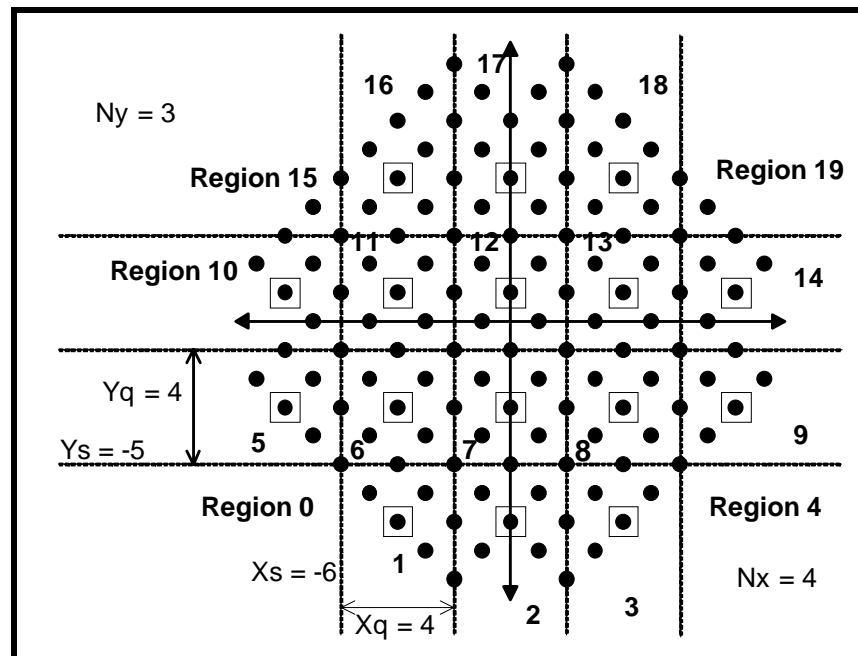


Figure 2. Dividing the Signal Space Diagram into Regions for 12 kbps – with Path-Index (Y_0 , Y_1 , and Y_2) = 000

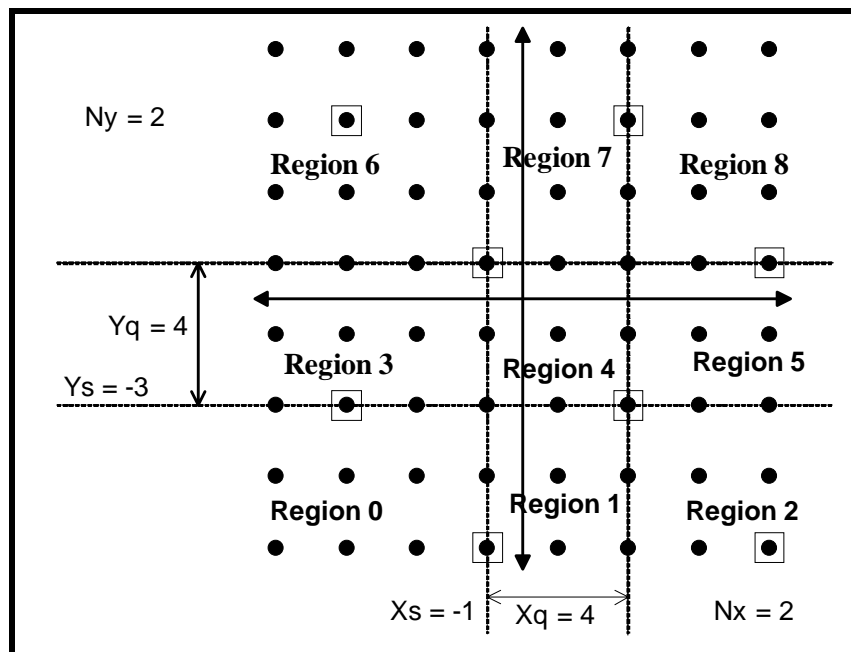


Figure 3. Dividing the Signal Space Diagram into Regions for 9.6 kbps – with Path-Index (Y_0 , Y_1 , and Y_2) = 000

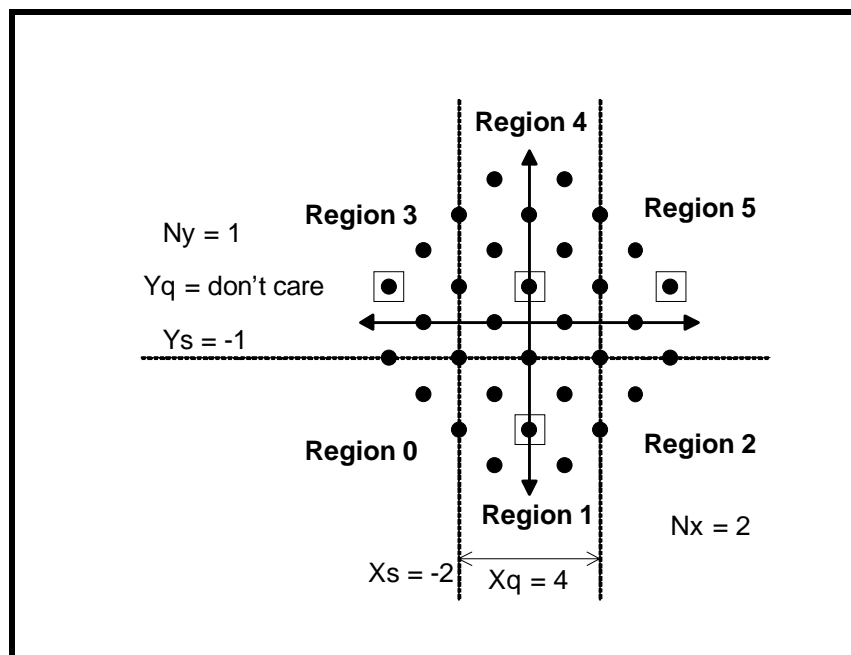
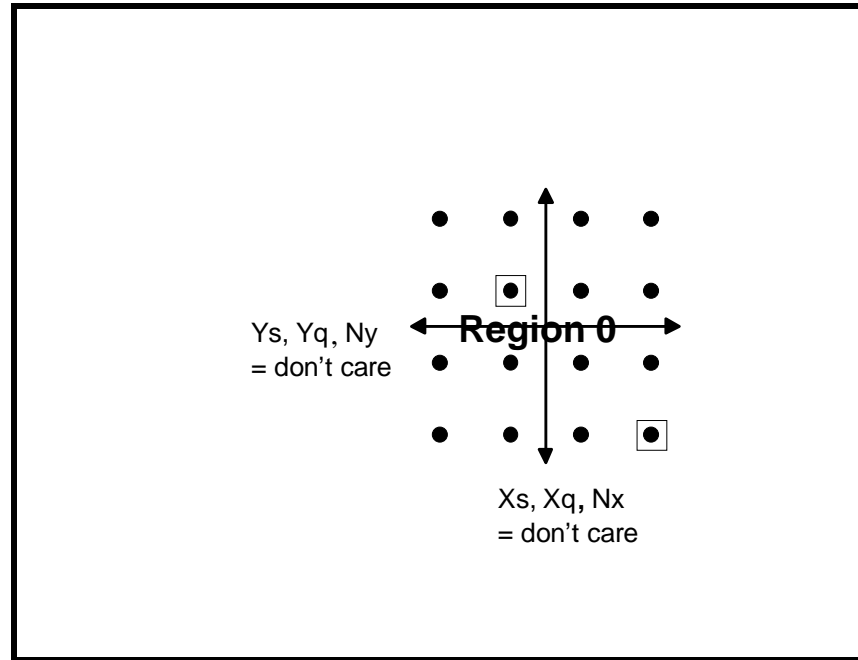


Figure 4. Dividing the Signal Space Diagram into Regions for 7.2 kbps – with Path-Index (Y_0 , Y_1 , and Y_2) = 000



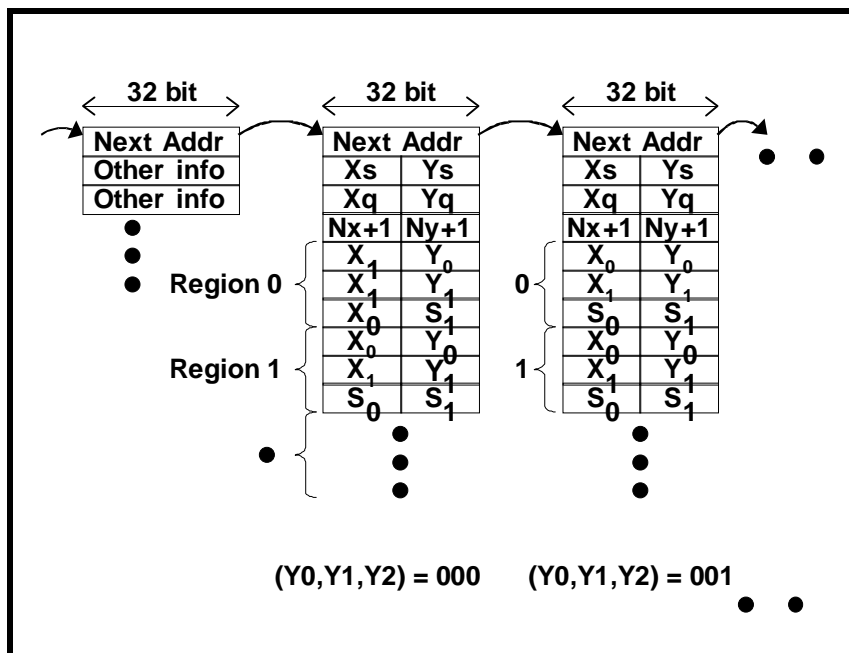
The diagrams show that once the region of the received symbol is determined, it is only necessary to calculate the distances from at most two path-states to find out which one is closest.

In the figures, X_s and Y_s represent the lowermost and leftmost X and Y locations of the dividing line. X_q and Y_q represent the X and Y distances between the dividing lines. N_x and N_y denote the number of vertical and horizontal dividing lines respectively. The regions are numbered from left to right and from bottom to top.

Finding the region is nothing more than a compare and increment step. Once the region is located, the region number can be used as an index to a table to look up the two closest path-states.

A look-up table to store the signal space diagram is created for each bit rate and for each point (Y_0 , Y_1 , and Y_2). The table is arranged as shown in Figure 5. The look-up table is arranged as a link-list to allow a variable sized table. Next, address points are set to the address of the next table for different (Y_0 , Y_1 , and Y_2) path indexes. Locations X_0 , Y_0 and X_1 , Y_1 represent the two path-states closest in distance to the received symbol, if the symbol is found within that particular region. If the region has only one possible closest path-state, the other X , Y values are marked with maximum numbers.

Figure 5. Structure of the Signal Space Constellation Diagram



To convert this step from C to C62xx assembly, the innermost two loops are completely unrolled and done in parallel. Only five cycles are needed to find the region number for a given received symbol location. The A and B register files of the C62xx are well suited to handle the X and Y dimensions respectively because the X and Y dimensions seldom interfere with each other. Therefore, minimum cross-path interaction is necessary.

Using this scheme, the memory size needed to store the signal space constellation look-up tables for all bit rates are listed in Table 2.

Table 2. Byte Size Requirement for Signal Space Constellation Look-up Table

Bit rate = 14.4 kbps	Look-up table byte size	
Next addr, Ys, Xs, Yq, Xq, Nx, Ny	16	
X ₀ , Y ₀ , X ₁ , Y ₁ , S ₀ , S ₁		12
Path-states for all regions		x20
For (Y ₀ ,Y ₁ ,Y ₂) = 000 to 111	x8	x8
Total		2048
Bit rate = 12 kbps	Look-up table byte size	
Next addr, Ys, Xs, Yq, Xq, Nx, Ny	16	
X ₀ , Y ₀ , X ₁ , Y ₁ , S ₀ , S ₁		12
Path-states for all regions		x9
For (Y ₀ ,Y ₁ ,Y ₂) = 000 to 111	x8	x8
Total		992
Bit rate = 9.6 kbps	Look-up table byte size	
Next addr, Ys, Xs, Yq, Xq, Nx, Ny	16	
X ₀ , Y ₀ , X ₁ , Y ₁ , S ₀ , S ₁		12
Path-states for all regions		x5
For (Y ₀ ,Y ₁ ,Y ₂) = 000 to 111	x8	x8
Total		608
Bit rate = 7.2 kbps	Look-up table byte size	
Next addr, Ys, Xs, Yq, Xq, Nx, Ny	16	
X ₀ , Y ₀ , X ₁ , Y ₁ , S ₀ , S ₁		12
Path-states for all regions		x1
For (Y ₀ ,Y ₁ ,Y ₂) = 000 to 111	x8	x8
Total		224

Calculating distances requires the square-root operation. To reduce the number of clock cycles, the square-root operation is not used when calculating the Euclidean distance. The received symbol X and Y location inputs are represented by a 16-bit signed value scaled up by 1024 (Q10). We use a 32-bit unsigned number to represent the distance. The actual meaning of the distance is not important as long as we can compare and accumulate its magnitude.



Step 3. Find Shortest Distances

C Code: Line 107 to 135
Assembly: Line 523 to 650

This step finds the smallest of the distances found in Step 2. The smallest of Distance[0] to Distance[4] is stored in Dist0 and the smallest of Distance[5] to Distance[7] is stored in Dist1. Based on which of the distances are smallest, two path-indexes are found. One is for $Y0 = 0$, the other is for $Y0 = 1$. The two corresponding path-states are stored in the DelayPath array. The two path-indexes are also fed into the TMBack look-up table to find the previous delay-state from the present delay-state.

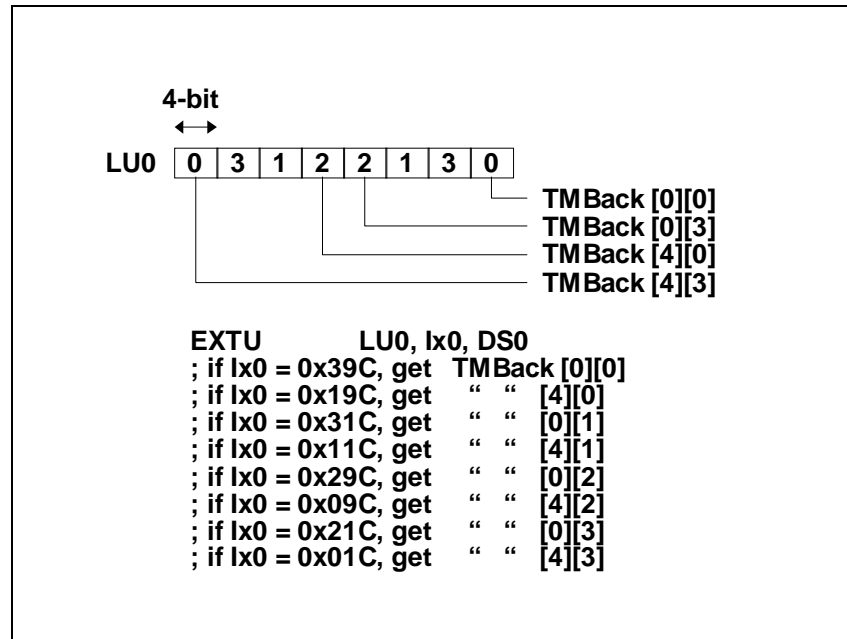
The outputs from this step are the previous delay-states stored in the delay-state registers DS0 through DS7. These will be used immediately by the next step. The same values are being stored in the DelayPath .bss section, pointed to by DStCurr. The data structure of the DelayPath array is shown in Figure 8 and is described in detail in *Step 5. Trace Backward for Path-State*.

To calculate the shortest distance, the loop is completely unrolled to reduce the branch overhead. Since all C62xx instructions can be conditional, it is easy to implement a simple if-then-else statement for the search algorithm, such as finding the maximum or minimum value.

To translate the two path-indexes to the previous delay-states, it is necessary to use the look-up table TMBack. The even rows of this look-up table translate the $Y0=0$ path-index to delay-states. The odd rows of this table translate the $Y0=1$ path-index to delay-states.

When it is necessary to look-up information in memory, the only method is to use the load instruction. However, the load instruction requires four cycles of delay slot. Using the look-up table method to find out the delay-state takes at least five cycles. Because the TMBack look-up table is small, we can read the complete table into the A or B register files to save cycles. Then, to look up, only a one cycle EXTU instruction is needed to locate the delay-state. Figure 6 shows how a C62xx 32-bit register can hold two rows of the TMBack look-up table. Since there are eight rows, only four C62xx registers are needed to hold the complete table.

Figure 6. Using Registers to Store the TMBBack Look-up Table



The content of the register is arranged in this manner to ease the task of reading the look-up table. For example, we can extract the upper 16 bits of LU0 instead of the lower 16 bits by toggling bit 9 of lx0 for the EXTU instruction. Thus, we can use lx0 to extract TMBBack[0][0]. Next, by setting bit 9 of lx0 to zero (this is done by a SUB 0x200 instruction), we can extract TMBBack[4][0].

Step 4. Calculate Accumulate Distances

C Code: Line 137 to 149

Assembly: Line 652 to 728

This step accumulates the Dist0 and Dist1 values into the AccDist array, which is an array of eight distances. The following formula summarizes this step.

$$\text{AccDist}[i] = \text{AccDist}[i] \times 7/8 + \text{Dist0} / 8; i = \text{even}$$

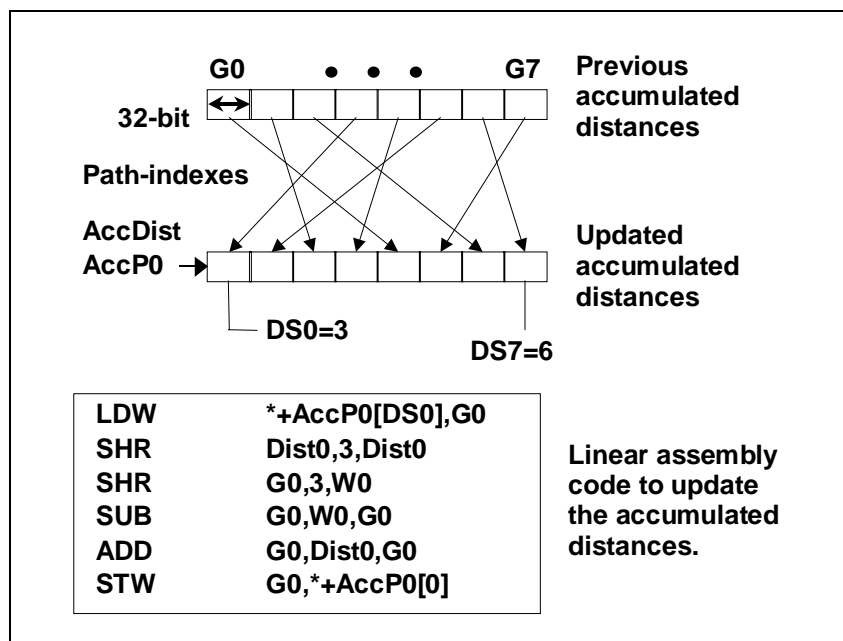
$$\text{AccDist}[i] = \text{AccDist}[i] \times 7/8 + \text{Dist1} / 8; i = \text{odd}$$

Using 7/8 and 1/8 as the constants in the above formula makes calculation simpler than using 9/10 or 1/10. This is because division by eight is a simple right shift by three times. To multiply by 7/8, we can subtract 1/8 of itself.

The AccDist array represents the total accumulated distances when tracing back from the current delay-states via the path-indexes. Therefore, it is necessary to obtain the previous accumulated distances using the previous delay-states DS0 to DS7 as the indexes. Fortunately, we have enough registers in the C62xx to store the AccDist array into the eight registers G0 to G7 first, before the distances are swapped.

Figure 7 shows an example of how this step works.

Figure 7. Calculating the Accumulated Distances





The outputs from this step are the eight accumulated distances stored in registers G0 through G7. These are used immediately by the next step. The same values are stored in the .bss section `AccDist`, which is used in the next pass of the Viterbi algorithm.

Step 5. Trace Backward for Path-State

C Code: Line 151 to 169

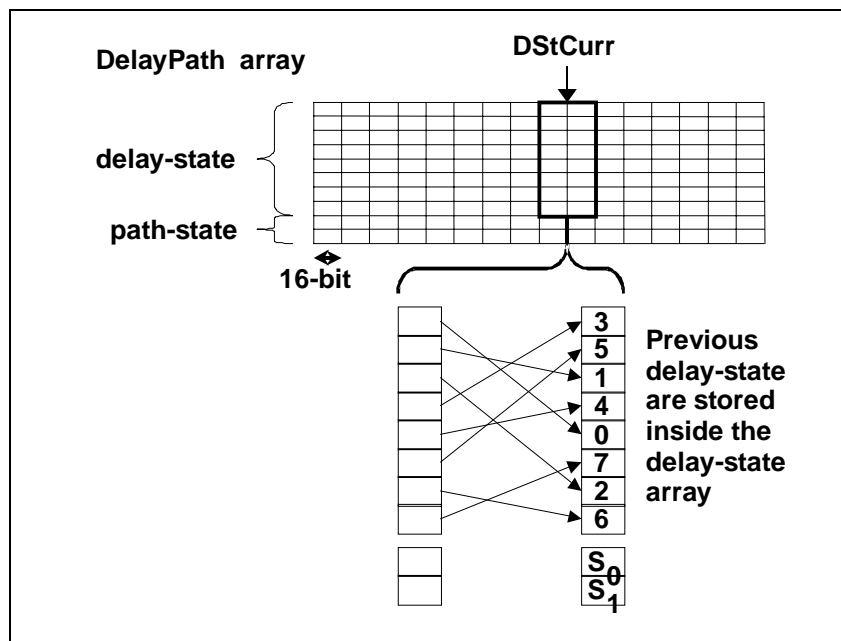
Assembly: Line 730 to 816

The accumulated distances G0 through G7, calculated from the previous Step 4, are used in

Step 5. The smallest of the accumulated distances must be found first. The delay-state of the smallest accumulated distance is then used as the starting point to trace fifteen passes backward. Once the delay-state fifteen passes backward is found, one of the two path-states stored in DelayPath array becomes the end result for this step. This result is contained in a temporary register TTT.

The DelayPath array is shown in Figure 8. Each element in this array is 16 bits wide. It contains 16 columns to represent a history of 16 passes of the Viterbi algorithm. The first eight elements of each column contain the location of the previous delay-states. The last two elements of each column contain the path-states, the path-indexes of which connect the present delay-states with the previous delay-states. DStCurr points to the present position. This step also increments the DStCurr so that it points to the next position in the array after each pass of the Viterbi algorithm. The final DStCurr value must be saved in a .bss section DstCurrbss so that it can be used after the function exits and reenters.

Figure 8. DelayPath Array Structure



DelayPath must be addressed in a circular fashion. Although the C62xx provides circular addressing capability, this implementation is not used because the byte size of this array is not a power of two. Implementation of circular addressing by compare and add/subtract does not add any clock cycle because that can be performed in parallel with other instructions.

The present delay-state with the smallest accumulate distance is placed in register IDD. Then a backward trace loop is done using circular addressing, as shown in the following linear assembly code.

```

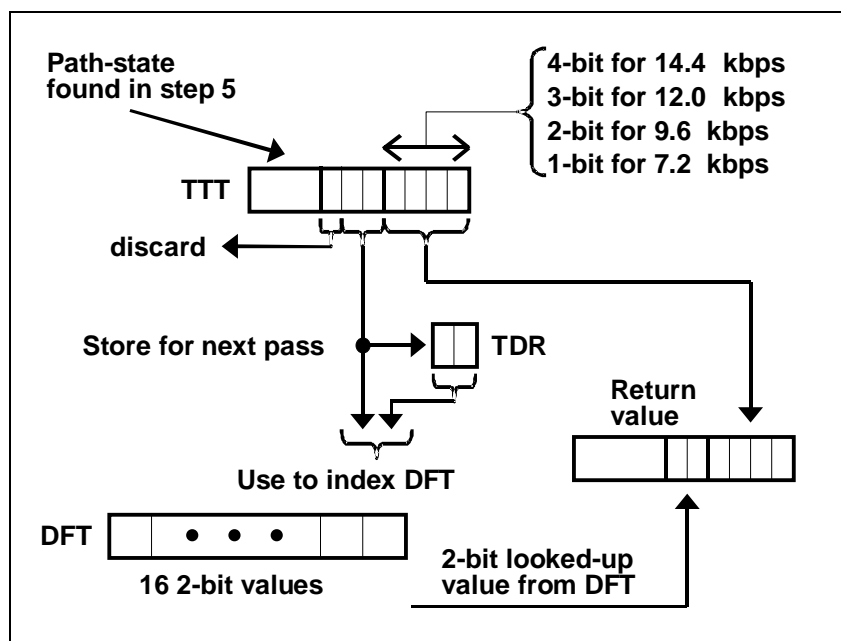
                                MVK      DelayPath,JJ0
                                MVKH     DelayPath,JJ0
                                MV        DStCurr,JJ
IILoop:                        MVK      15,II
                                LDH       *+JJ[IDD],IDD
                                CMPGT     JJ,JJ0,TT0
                                [TT0]     SUBAH  JJ,10,JJ
                                [!TT0]    MVK     DelayPath+300,JJ
                                [!TT0]    MVKH     DelayPath+300,JJ
                                SUB       II,1,II
                                [II]      B       IILoop
```

Step 6. Differentiate

C Code: Line 171 to 177
 Assembly: Line 818 to 865

Step 6 uses the path-state TTT (from Step 5) as the input. For a bit rate of 14.4 kbps, the lower four bits are directly passed to the function return register. The other three bits are passed to the Diff look-up table to retrieve a differentiated output. Figure 9 shows how this step works.

Figure 9. Differentiation



TDR is used to store the bits from the previous pass of the Viterbi algorithm. The TDR bits, combined with the present bits from the TTT registers, are used to address the DFT array.

Since the Diff look-up table is also small, we can use the same scheme as shown in *Step 3. Find Shortest Distances*, to save cycles when reading look-up tables. The whole Diff table is first read into a 32-bit register DFT. Then, using appropriate shift and bit-field commands of the C62xx, it is possible to read the look-up table in one cycle. The result is passed to register A4 as the return value.



Step 7. Closing the Function

C Code: Line 179 to 180
Assembly: Line 867 to 870

The return sequence restores the registers from the stack and jumps back to the calling routine. Note that, because of the pipeline structure of the code, a large portion of this step is combined with the previous Step 6 in a parallel fashion. Documentation using “asterisks” in the comment field of each instruction helps the reader know which instruction belongs to each step.



Building and Testing the Code

The C62xx simulator can be used to count the number of cycles required to execute a portion of the code section. That way, it is possible to estimate the total time to execute the Viterbi algorithm. The current version of the simulator includes the time required for cache access, external memory access, memory bank conflicts, and memory stall. To accurately estimate the time used by the Viterbi algorithm, the Viterbi function is placed in internal program RAM and the lookup tables are placed in internal data RAM using appropriate linker command files.

The complete program, including the test code, is made up of the following source files:

- ❑ vectors.asm: jumps to c_int00
- ❑ main.h: definition for main.c
- ❑ main.c: calling routines to test the Viterbi algorithms
- ❑ lookup.c: lookup tables for signal space constellation
- ❑ viterbic.c: Viterbi implementation in C code
- ❑ viterbi.asm: Viterbi implementation in C62xx assembly code
- ❑ v.cmd Linker command file for assembly implementation
- ❑ vv.cmd Linker command file for C implementation
- ❑ c.bat Compile, assemble, link for assembly implementation
- ❑ cc.bat Compile, assemble, link for C implementation

The main.c files contains the following module:

```
void Trellis (int DataIn, int *Xo, int *Yo);
```

The purpose of this routine is to generate the symbol X and Y location to be sent to the Viterbi algorithm for testing.

```
int ReadData ();
```

This routine reads a list of test data.

```
void OutData (int DataOut);
```

This routine outputs the decoded result.

```
main ();
```

This routine runs the test to the Viterbi code. The following lists a simplified section of the main routine.

```
main ()
{
    int DataIn, DataOut, Xo, Yo, Xi, Yi;

    Init ();
    for (;;)
    {
        DataIn = ReadData();
        Trellis (DataIn, &Xo, &Yo);
        Xi = Xo;  Yi = Yo;
        ● DataOut = Viterbi (Xi,Yi,Mod14);
        ● OutData (DataOut);
    }
}
```

The “●” symbols in the above listing indicate the locations where breakpoints are placed for benchmarking. Between each, the RUNB command was used to test out the number of clock cycles. Table 3 summarizes the results.

Table 3. Clock Cycle Requirements for V.32bis Viterbi Decoding Algorithm.

Implementation Method	Cycles
C code	6612
C62xx assembly	329

The cycle count for the C code is without optimization. With proper optimization, it is possible to achieve reduced cycle count even when using C implementation. The Viterbi C code listed in this application report is demonstrating only how the assembly code functions.



Conclusion

Using the assembly implementation, at 2400 symbols/sec, the percentage loading of a 200 MHz C62xx is equal to

$$329 \times 2400 / 200,000,000 = 0.39\%$$

This report describes the implementation of the V.32bis Viterbi decoding algorithm on the C62xxDSP. Implementation of this algorithm on the C62xx shows that, although the C62xx assembly instructions are very easy to read and understand, it is best to write a complementary C program for good documentation purposes. The programmer can then tune the C program using the techniques described in the *TMS320C62x/C67x Programmer's Guide*. This will eventually improve the execution time while maintaining good documentation.

Appendix A. V.32BIS Viterbi C Code Implementation

```

1  /*****
2  * File:      viterbic.c
3  * Purpose:   Viterbi in C code.
4  *****/
5
6  #include "main.h"
7
8  /* Differentiator truth table. Bit 4,5 for transmitter, bit 0,1 receiver. */
9  char Diff[] = { 0x00, 0x11, 0x23, 0x32, 0x11, 0x00, 0x32, 0x23,
10                  0x22, 0x33, 0x10, 0x01, 0x33, 0x22, 0x01, 0x10 };
11
12  /* Delay states transition matrix looking backward from present state. */
13  char TMBackward[8][4] = {
14      0, 3, 1, 2,
15      4, 7, 6, 5,
16      1, 2, 0, 3,
17      5, 6, 7, 4,
18      2, 1, 3, 0,
19      7, 4, 5, 6,
20      3, 0, 2, 1,
21      6, 5, 4, 7
22  };
23
24  char Tdr, DelayState[16][8], PathState[16][2], DStCurr;
25  int AccDist[8];
26
27  /*****
28  * Init - initialization.
29  *****/
30
31  void Init ()
32  {
33      short i, j;
34      DStCurr = 0;
35      Tdr = 0;
36      AccDist[0] = 0;
37      for (i = 1; i < 8; i++)
38          AccDist[i] = (1 << 10) / 2;
39      for (j = 0; j < 16; j++)
40          for (i = 0; i < 8; i++)
41              DelayState[j][i] = 0;
42      for (j = 0; j < 16; j++)
43          for (i = 0; i < 2; i++)
44              PathState[j][i] = 0;
45  }
46
47  /*****
48  * Viterbi - decoder.
49  *****/
50
51  int Viterbi (int Xi, int Yi, int modd)
52  {
53      short State[8], Xb, Yb, Nx, Ny, Qx, Qy, Ix, Iy, Id;
54      short c, i, j, tmp1, tmp2, tmp3, tmp4, tt, ss, pp;
55      int Distance[8], dist1, dist0, DataOut, Sht;
56      Mod *Mcurr;
57
58      Mcurr = (Mod*) modd;
59      Sht = Mcurr->TSht;
60
61      /* With Xi and Yi as input, first get the shortest distances */
62
63      for (c = 0; c < 8; c++)

```



```

64  {
65      Xb = (Mcurr->TXs[c]) << 10;  Yb = (Mcurr->TYs[c]) << 10;
66      Nx = Mcurr->TNx[c];           Ny = Mcurr->TNy[c];
67      Qx = (Mcurr->TQx) << 10;      Qy = (Mcurr->TQy) << 10;
68      Iy = Ny;
69      for (i = 0; i < Ny; i++)
70      {
71          if (Yi < Yb) { Iy = i; break; }
72          Yb += Qy;
73      }
74      Ix = Nx;
75      for (i = 0; i < Nx; i++)
76      {
77          if (Xi < Xb) { Ix = i; break; }
78          Xb += Qx;
79      }
80      Id = Iy * (Nx + 1) + Ix;
81      State[c] = *(Mcurr->St[c] + Id);
82      tmp1 = *(Mcurr->Xc[c] + Id);
83      tmp2 = *(Mcurr->Yc[c] + Id);
84      if (State[c] != SS)
85      {
86          tmp3 = Xi - (tmp1 << 10);
87          tmp4 = Yi - (tmp2 << 10);
88          Distance[c] = (int)tmp3 * (int)tmp3 + (int)tmp4 * (int)tmp4;
89      }
90      else
91      {
92          tmp3 = Xi - (*(Mcurr->Xc[c] + tmp1) << 10);
93          tmp4 = Yi - (*(Mcurr->Yc[c] + tmp1) << 10);
94          dist1 = (int)tmp3 * (int)tmp3 + (int)tmp4 * (int)tmp4;
95          tmp3 = Xi - (*(Mcurr->Xc[c] + tmp2) << 10);
96          tmp4 = Yi - (*(Mcurr->Yc[c] + tmp2) << 10);
97          Distance[c] = (int)tmp3 * (int)tmp3 + (int)tmp4 * (int)tmp4;
98          State[c] = *(Mcurr->St[c] + tmp2);
99          if (dist1 < Distance[c])
100         {
101             Distance[c] = dist1;
102             State[c] = *(Mcurr->St[c] + tmp1);
103         }
104     }
105 }
106
107 /* Find the shortest distance among path state 0-3 and 4-7 */
108
109 dist0 = 0x7FFFFFFF;
110 for (c = 0; c < 4; c++)
111 {
112     if (Distance[c] < dist0)
113     {
114         dist0 = Distance[c];
115         Id = c;
116     } }
117 DelayState[DStCurr][0] = TMBbackward[0][Id];
118 DelayState[DStCurr][2] = TMBbackward[2][Id];
119 DelayState[DStCurr][4] = TMBbackward[4][Id];
120 DelayState[DStCurr][6] = TMBbackward[6][Id];
121 PathState[DStCurr][0] = State[Id];
122
123 dist1 = 0x7FFFFFFF;
124 for (c = 4; c < 8; c++)
125 {
126     if (Distance[c] < dist1)
127     {
128         dist1 = Distance[c];
129         Id = c;
130     } }

```

```
131 DelayState[DStCurr][1] = TMBBackward[1][Id-4];
132 DelayState[DStCurr][3] = TMBBackward[3][Id-4];
133 DelayState[DStCurr][5] = TMBBackward[5][Id-4];
134 DelayState[DStCurr][7] = TMBBackward[7][Id-4];
135 PathState[DStCurr][1] = State[Id];
136
137 /* Update accumulated distance */
138
139 for (i = 0; i < 8; i++)
140     Distance[i] = AccDist[i];
141 for (i = 0; i < 8; i++)
142 {
143     Id = DelayState[DStCurr][i];
144     AccDist[i] = Distance[Id] / 8 * 7;
145 }
146 for (i = 0; i < 8; i+=2)
147     AccDist[i] += dist0 / 8;
148 for (i = 1; i < 8; i+=2)
149     AccDist[i] += dist1 / 8;
150
151 /* Trace backward to get output */
152
153 dist0 = 0x7FFFFFFF;
154 for (i = 0; i < 8; i++)
155 {
156     if (AccDist[i] < dist0)
157     {
158         dist0 = AccDist[i];
159         Id = i;
160     } }
161 j = DStCurr;
162 for (i = 0; i < 15; i++)
163 {
164     Id = DelayState[j][Id];
165     j--; if (j < 0) j = 15;
166 }
167 tt = PathState[j][Id % 2];
168 DStCurr++;
169 if (DStCurr >= 16) DStCurr = 0;
170
171 /* Differentiate */
172
173 ss = tt & ((1 << Sht) - 1);
174 tt = (tt >> Sht) & 0x03;
175 pp = (tt << 2) | Tdr;
176 Tdr = tt;
177 DataOut = ((Diff[pp] & 0x03) << Sht) | ss;
178
179 return (DataOut);
180 }
181
182 /***** End *****/
```



Appendix B. V.32Bis Viterbi C62XX Assembly Implementation

```

1 *****
2 *
3 *   TEXAS INSTRUMENTS, INC.
4 *
5 *   V.32BIS VITERBI DECODING
6 *
7 *   FILE:   Viterbi.asm
8 *
9 *   REVISION DATE:   12/08/97
10 *
11 *   USAGE: This routine is C callable and can be called as
12 *
13 *       Data = int Viterbi (int Xi, int Yi, int* Mod);
14 *
15 *       Xi = Symbol location in "real" axis, scaled up by 1024.
16 *       Yi = Symbol location in "imaginary" axis, scaled up by 1024.
17 *       Mod = Modulation table to use.
18 *       Data = return symbol data value.
19 *
20 *   DESCRIPTION:  A complementary C code and a complete calling example code
21 *                  should be included with this file.
22 *
23 *****
24
25 *****
26 * Variables declaration.
27 *****
28
29 ***** Global and static variables:
30         .bss      Tdr,4,4           ; Differentiator Tdr value.
31         .bss      DStCurrbss,4,4     ; Current delay state pointer.
32         .bss      DelayPath,320,4    ; DelayState and PathState array in half.
33         .bss      AccDist,32,4       ; Accumulated distance array in word.
34
35 ***** Variables that are temporary and local:
36         .bss      Distance,32,4      ; Distance Result array in word.
37         .bss      Dummy,2,2          ; To allow simultaneous access.
38         .bss      State,16,4         ; State array in half.
39         .bss      MCurr,4,4          ; Save current modulation pointer.
40
41 ***** Constant declaration:
42 SK      .set      1024               ; Scale factor.
43
44 *****
45 * Lookup Tables.
46 *****
47
48         .sect     "LUTable"
49
50 ***** Modulation tables:
51
52 MMMMM   .set      0x7FFF             ; Set to maximum (for modulation table use)
53 SS      .set      0xFFFF             ; Unused state value (for mod table use)
54 SZ      .set      3                  ; Data point word size (for mod table use)
55
56         .def      _M14               ; Make this table visible to outside.
57 _M14:    ; Modulation table for 14.4 kbps.
58         .int      M140               ; First address.
59         .int      0x35E              ; Extract bit 4 - 5 (The Y bits in V.32bis)
60         .int      0x39C              ; Extract bit 0 - 3 (The Q bits in V.32bis)
61         .int      4                  ; Shift amount (Number of Q bits in V.32bis)
62

```



```
63 M140 .int M141 ; Next address
64 .short -5*SK, -6*SK, 4*SK, 4*SK, (3+1)*SZ, (4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
65 .short -7*SK, -4*SK, -3*SK, -8*SK, 0x05, 0x00 ; Y0,X0,Y1,X1,S0,S1
66 .short -7*SK, -4*SK, MMMMM, MMMMM, 0x05, SS
67 .short -7*SK, +0*SK, MMMMM, MMMMM, 0x07, SS
68 .short -7*SK, +4*SK, MMMMM, MMMMM, 0x03, SS
69 .short -7*SK, +4*SK, -3*SK, +8*SK, 0x03, 0x01
70 .short -3*SK, -8*SK, MMMMM, MMMMM, 0x00, SS
71 .short -3*SK, -4*SK, MMMMM, MMMMM, 0x04, SS
72 .short -3*SK, +0*SK, MMMMM, MMMMM, 0x06, SS
73 .short -3*SK, +4*SK, MMMMM, MMMMM, 0x02, SS
74 .short -3*SK, +8*SK, MMMMM, MMMMM, 0x01, SS
75 .short +1*SK, -8*SK, MMMMM, MMMMM, 0x08, SS
76 .short +1*SK, -4*SK, MMMMM, MMMMM, 0x0C, SS
77 .short +1*SK, +0*SK, MMMMM, MMMMM, 0x0E, SS
78 .short +1*SK, +4*SK, MMMMM, MMMMM, 0x0A, SS
79 .short +1*SK, +8*SK, MMMMM, MMMMM, 0x09, SS
80 .short +5*SK, -4*SK, +1*SK, -8*SK, 0x0D, 0x08
81 .short +5*SK, -4*SK, MMMMM, MMMMM, 0x0D, SS
82 .short +5*SK, +0*SK, MMMMM, MMMMM, 0x0F, SS
83 .short +5*SK, +4*SK, MMMMM, MMMMM, 0x0B, SS
84 .short +5*SK, +4*SK, +1*SK, +8*SK, 0x0B, 0x09
85
86 M141 .int M142 ; Next address
87 .short -3*SK, -6*SK, 4*SK, 4*SK, (3+1)*SZ, (4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
88 .short -5*SK, -4*SK, -1*SK, -8*SK, 0x1B, 0x19 ; Y0,X0,Y1,X1,S0,S1
89 .short -5*SK, -4*SK, MMMMM, MMMMM, 0x1B, SS
90 .short -5*SK, +0*SK, MMMMM, MMMMM, 0x1F, SS
91 .short -5*SK, +4*SK, MMMMM, MMMMM, 0x1D, SS
92 .short -5*SK, +4*SK, -1*SK, +8*SK, 0x1D, 0x18
93 .short -1*SK, -8*SK, MMMMM, MMMMM, 0x19, SS
94 .short -1*SK, -4*SK, MMMMM, MMMMM, 0x1A, SS
95 .short -1*SK, +0*SK, MMMMM, MMMMM, 0x1E, SS
96 .short -1*SK, +4*SK, MMMMM, MMMMM, 0x1C, SS
97 .short -1*SK, +8*SK, MMMMM, MMMMM, 0x18, SS
98 .short +3*SK, -8*SK, MMMMM, MMMMM, 0x11, SS
99 .short +3*SK, -4*SK, MMMMM, MMMMM, 0x12, SS
100 .short +3*SK, +0*SK, MMMMM, MMMMM, 0x16, SS
101 .short +3*SK, +4*SK, MMMMM, MMMMM, 0x14, SS
102 .short +3*SK, +8*SK, MMMMM, MMMMM, 0x10, SS
103 .short +7*SK, -4*SK, +3*SK, -8*SK, 0x13, 0x11
104 .short +7*SK, -4*SK, MMMMM, MMMMM, 0x13, SS
105 .short +7*SK, +0*SK, MMMMM, MMMMM, 0x17, SS
106 .short +7*SK, +4*SK, MMMMM, MMMMM, 0x15, SS
107 .short +7*SK, +4*SK, +3*SK, +8*SK, 0x15, 0x10
108
109 M142 .int M143 ; Next address
110 .short -7*SK, -4*SK, 4*SK, 4*SK, (4+1)*SZ, (3+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
111 .short -9*SK, -2*SK, -5*SK, -6*SK, 0x28, 0x2D ; Y0,X0,Y1,X1,S0,S1
112 .short -9*SK, -2*SK, MMMMM, MMMMM, 0x28, SS
113 .short -9*SK, +2*SK, MMMMM, MMMMM, 0x20, SS
114 .short -9*SK, +2*SK, -5*SK, +6*SK, 0x20, 0x25
115 .short -5*SK, -6*SK, MMMMM, MMMMM, 0x2D, SS
116 .short -5*SK, -2*SK, MMMMM, MMMMM, 0x2C, SS
117 .short -5*SK, +2*SK, MMMMM, MMMMM, 0x24, SS
118 .short -5*SK, +6*SK, MMMMM, MMMMM, 0x25, SS
119 .short -1*SK, -6*SK, MMMMM, MMMMM, 0x2F, SS
120 .short -1*SK, -2*SK, MMMMM, MMMMM, 0x2E, SS
121 .short -1*SK, +2*SK, MMMMM, MMMMM, 0x26, SS
122 .short -1*SK, +6*SK, MMMMM, MMMMM, 0x27, SS
123 .short +3*SK, -6*SK, MMMMM, MMMMM, 0x2B, SS
124 .short +3*SK, -2*SK, MMMMM, MMMMM, 0x2A, SS
125 .short +3*SK, +2*SK, MMMMM, MMMMM, 0x22, SS
126 .short +3*SK, +6*SK, MMMMM, MMMMM, 0x23, SS
127 .short +7*SK, -2*SK, +3*SK, -6*SK, 0x29, 0x2B
128 .short +7*SK, -2*SK, MMMMM, MMMMM, 0x29, SS
129 .short +7*SK, +2*SK, MMMMM, MMMMM, 0x21, SS
```




```

130      .short  +7*SK,+2*SK,+3*SK,+6*SK,0x21,0x23
131
132 M143      .int    M144                                ; Next address
133      .short  -5*SK,-4*SK,4*SK,4*SK,(4+1)*SZ,(3+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
134      .short  -7*SK,-2*SK,-3*SK,-6*SK,0x31,0x33      ; Y0,X0,Y1,X1,S0,S1
135      .short  -7*SK,-2*SK,MMMMM,MMMMM,0x31,SS
136      .short  -7*SK,+2*SK,MMMMM,MMMMM,0x39,SS
137      .short  -7*SK,+2*SK,-3*SK,+6*SK,0x39,0x3B
138      .short  -3*SK,-6*SK,MMMMM,MMMMM,0x33,SS
139      .short  -3*SK,-2*SK,MMMMM,MMMMM,0x32,SS
140      .short  -3*SK,+2*SK,MMMMM,MMMMM,0x3A,SS
141      .short  -3*SK,+6*SK,MMMMM,MMMMM,0x3B,SS
142      .short  +1*SK,-6*SK,MMMMM,MMMMM,0x37,SS
143      .short  +1*SK,-2*SK,MMMMM,MMMMM,0x36,SS
144      .short  +1*SK,+2*SK,MMMMM,MMMMM,0x3E,SS
145      .short  +1*SK,+6*SK,MMMMM,MMMMM,0x3F,SS
146      .short  +5*SK,-6*SK,MMMMM,MMMMM,0x35,SS
147      .short  +5*SK,-2*SK,MMMMM,MMMMM,0x34,SS
148      .short  +5*SK,+2*SK,MMMMM,MMMMM,0x3C,SS
149      .short  +5*SK,+6*SK,MMMMM,MMMMM,0x3D,SS
150      .short  +9*SK,-2*SK,+5*SK,-6*SK,0x30,0x35
151      .short  +9*SK,-2*SK,MMMMM,MMMMM,0x30,SS
152      .short  +9*SK,+2*SK,MMMMM,MMMMM,0x38,SS
153      .short  +9*SK,+2*SK,+5*SK,+6*SK,0x38,0x3D
154
155 M144      .int    M145                                ; Next address
156      .short  -4*SK,-5*SK,4*SK,4*SK,(3+1)*SZ,(4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
157      .short  -6*SK,-3*SK,-2*SK,-7*SK,0x4B,0x49      ; Y0,X0,Y1,X1,S0,S1
158      .short  -6*SK,-3*SK,MMMMM,MMMMM,0x4B,SS
159      .short  -6*SK,+1*SK,MMMMM,MMMMM,0x4F,SS
160      .short  -6*SK,+5*SK,MMMMM,MMMMM,0x4D,SS
161      .short  -6*SK,+5*SK,-2*SK,+9*SK,0x4D,0x48
162      .short  -2*SK,-7*SK,MMMMM,MMMMM,0x49,SS
163      .short  -2*SK,-3*SK,MMMMM,MMMMM,0x4A,SS
164      .short  -2*SK,+1*SK,MMMMM,MMMMM,0x4E,SS
165      .short  -2*SK,+5*SK,MMMMM,MMMMM,0x4C,SS
166      .short  -2*SK,+9*SK,MMMMM,MMMMM,0x48,SS
167      .short  +2*SK,-7*SK,MMMMM,MMMMM,0x41,SS
168      .short  +2*SK,-3*SK,MMMMM,MMMMM,0x42,SS
169      .short  +2*SK,+1*SK,MMMMM,MMMMM,0x46,SS
170      .short  +2*SK,+5*SK,MMMMM,MMMMM,0x44,SS
171      .short  +2*SK,+9*SK,MMMMM,MMMMM,0x40,SS
172      .short  +6*SK,-3*SK,+2*SK,-7*SK,0x43,0x41
173      .short  +6*SK,-3*SK,MMMMM,MMMMM,0x43,SS
174      .short  +6*SK,+1*SK,MMMMM,MMMMM,0x47,SS
175      .short  +6*SK,+5*SK,MMMMM,MMMMM,0x45,SS
176      .short  +6*SK,+5*SK,+2*SK,+9*SK,0x45,0x40
177
178 M145      .int    M146                                ; Next address
179      .short  -4*SK,-7*SK,4*SK,4*SK,(3+1)*SZ,(4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
180      .short  -6*SK,-5*SK,-2*SK,-9*SK,0x55,0x50      ; Y0,X0,Y1,X1,S0,S1
181      .short  -6*SK,-5*SK,MMMMM,MMMMM,0x55,SS
182      .short  -6*SK,-1*SK,MMMMM,MMMMM,0x57,SS
183      .short  -6*SK,+3*SK,MMMMM,MMMMM,0x53,SS
184      .short  -6*SK,+3*SK,-2*SK,+7*SK,0x53,0x51
185      .short  -2*SK,-9*SK,MMMMM,MMMMM,0x50,SS
186      .short  -2*SK,-5*SK,MMMMM,MMMMM,0x54,SS
187      .short  -2*SK,-1*SK,MMMMM,MMMMM,0x56,SS
188      .short  -2*SK,+3*SK,MMMMM,MMMMM,0x52,SS
189      .short  -2*SK,+7*SK,MMMMM,MMMMM,0x51,SS
190      .short  +2*SK,-9*SK,MMMMM,MMMMM,0x58,SS
191      .short  +2*SK,-5*SK,MMMMM,MMMMM,0x5C,SS
192      .short  +2*SK,-1*SK,MMMMM,MMMMM,0x5E,SS
193      .short  +2*SK,+3*SK,MMMMM,MMMMM,0x5A,SS
194      .short  +2*SK,+7*SK,MMMMM,MMMMM,0x59,SS
195      .short  +6*SK,-5*SK,+2*SK,-9*SK,0x5D,0x58
196      .short  +6*SK,-5*SK,MMMMM,MMMMM,0x5D,SS

```



```
197      .short  +6*SK, -1*SK, MMMMM, MMMMM, 0x5F, SS
198      .short  +6*SK, +3*SK, MMMMM, MMMMM, 0x5B, SS
199      .short  +6*SK, +3*SK, +2*SK, +7*SK, 0x5B, 0x59
200
201 M146      .int    M147                                ; Next address
202      .short  -6*SK, -5*SK, 4*SK, 4*SK, (4+1)*SZ, (3+1)*SZ ; Ys, Xs, Yq, Xq, Ny+1, Nx+1
203      .short  -8*SK, -3*SK, -4*SK, -7*SK, 0x61, 0x63      ; Y0, X0, Y1, X1, S0, S1
204      .short  -8*SK, -3*SK, MMMMM, MMMMM, 0x61, SS
205      .short  -8*SK, +1*SK, MMMMM, MMMMM, 0x69, SS
206      .short  -8*SK, +1*SK, -4*SK, +5*SK, 0x69, 0x6B
207      .short  -4*SK, -7*SK, MMMMM, MMMMM, 0x63, SS
208      .short  -4*SK, -3*SK, MMMMM, MMMMM, 0x62, SS
209      .short  -4*SK, +1*SK, MMMMM, MMMMM, 0x6A, SS
210      .short  -4*SK, +5*SK, MMMMM, MMMMM, 0x6B, SS
211      .short  +0*SK, -7*SK, MMMMM, MMMMM, 0x67, SS
212      .short  +0*SK, -3*SK, MMMMM, MMMMM, 0x66, SS
213      .short  +0*SK, +1*SK, MMMMM, MMMMM, 0x6E, SS
214      .short  +0*SK, +5*SK, MMMMM, MMMMM, 0x6F, SS
215      .short  +4*SK, -7*SK, MMMMM, MMMMM, 0x65, SS
216      .short  +4*SK, -3*SK, MMMMM, MMMMM, 0x64, SS
217      .short  +4*SK, +1*SK, MMMMM, MMMMM, 0x6C, SS
218      .short  +4*SK, +5*SK, MMMMM, MMMMM, 0x6D, SS
219      .short  +8*SK, -3*SK, +4*SK, -7*SK, 0x60, 0x65
220      .short  +8*SK, -3*SK, MMMMM, MMMMM, 0x60, SS
221      .short  +8*SK, +1*SK, MMMMM, MMMMM, 0x68, SS
222      .short  +8*SK, +1*SK, +4*SK, +5*SK, 0x68, 0x6D
223
224 M147      .int    M140                                ; Next address
225      .short  -6*SK, -3*SK, 4*SK, 4*SK, (4+1)*SZ, (3+1)*SZ ; Ys, Xs, Yq, Xq, Ny+1, Nx+1
226      .short  -8*SK, -1*SK, -4*SK, -5*SK, 0x78, 0x7D      ; Y0, X0, Y1, X1, S0, S1
227      .short  -8*SK, -1*SK, MMMMM, MMMMM, 0x78, SS
228      .short  -8*SK, +3*SK, MMMMM, MMMMM, 0x70, SS
229      .short  -8*SK, +3*SK, -4*SK, +7*SK, 0x70, 0x75
230      .short  -4*SK, -5*SK, MMMMM, MMMMM, 0x7D, SS
231      .short  -4*SK, -1*SK, MMMMM, MMMMM, 0x7C, SS
232      .short  -4*SK, +3*SK, MMMMM, MMMMM, 0x74, SS
233      .short  -4*SK, +7*SK, MMMMM, MMMMM, 0x75, SS
234      .short  +0*SK, -5*SK, MMMMM, MMMMM, 0x7F, SS
235      .short  +0*SK, -1*SK, MMMMM, MMMMM, 0x7E, SS
236      .short  +0*SK, +3*SK, MMMMM, MMMMM, 0x76, SS
237      .short  +0*SK, +7*SK, MMMMM, MMMMM, 0x77, SS
238      .short  +4*SK, -5*SK, MMMMM, MMMMM, 0x7B, SS
239      .short  +4*SK, -1*SK, MMMMM, MMMMM, 0x7A, SS
240      .short  +4*SK, +3*SK, MMMMM, MMMMM, 0x72, SS
241      .short  +4*SK, +7*SK, MMMMM, MMMMM, 0x73, SS
242      .short  +8*SK, -1*SK, +4*SK, -5*SK, 0x79, 0x7B
243      .short  +8*SK, -1*SK, MMMMM, MMMMM, 0x79, SS
244      .short  +8*SK, +3*SK, MMMMM, MMMMM, 0x71, SS
245      .short  +8*SK, +3*SK, +4*SK, +7*SK, 0x71, 0x73
246
247      .def      _M12                                ; Make this table visible to outside.
248 _M12:      ; Modulation table for 12.0 kbps.
249      ; . . .
250      ; . . .
251
252 ***** Transition matrix backward table:
253
254 TMBack      .int    0x03122130                      ; 2D array index:  43 42 41 40 03 02 01 00
255      .int    0x65475674                      ; (data in 4-bit)  53 52 51 50 13 12 11 10
256      .int    0x12033021                      ;                63 62 61 60 23 22 21 20
257      .int    0x74564765                      ;                73 72 71 70 33 32 31 30
258
259 ***** Receiver differentiator table:
260
261 Diff      .int    0x1B4EE1B4                      ; index (data in 2-bit):  F E D C ... 2 1 0
262
263 *****
```



```

264 * Initialization of the variables.
265 *****
266
267         .text
268         .def      _Init
269 _Init:
270
271 ***** DStCurrbss = *DelayPath.
272         MVK      DStCurrbss,A0
273         MVKH     DStCurrbss,A0
274         MVK      DelayPath,A1
275         MVKH     DelayPath,A1
276         STW      A1,*A0
277
278 ***** DelayPath[i][j] = 0, i from 0 to 15, j from 0 to 9.
279         MVK      16,B0
280 PLoop:
281         MVK      10,B1
282         ZERO     A0
283 KLoop:
284         SUB      10,B1,A3
285         STH      A0,*+A1[A3]
286 [B1] SUB      B1,1,B1
287 [B1] B          KLoop
288         NOP      5
289         ADDAH    A1,10,A1
290 [B0] SUB      B0,1,B0
291 [B0] B          PLoop
292         NOP      5
293
294 ***** AccDist[i] = 512, i from 1 to 7.
295 ***** AccDist[0] = 0.
296         MVK      SK/2,A0
297         MVK      AccDist,A1
298         MVKH     AccDist,A1
299         MVK      8,B0
300 ILoop:
301         SUB      8,B0,A3
302         STW      A0,*+A1[A3]
303 [B0] SUB      B0,1,B0
304 [B0] B          ILoop
305         NOP      5
306         ZERO     A0
307         STW      A0,*+A1[0]
308
309 ***** Tdr = 0.
310         ZERO     A0
311         MVK      Tdr,A1
312         MVKH     Tdr,A1
313         STW      A0,*A1
314
315 ***** Return.
316         B        B3
317         NOP      5
318
319 *****
320 * Viterbi Program section.
321 *****
322
323 *** Registers name that will be used for all the Steps.
324 DStCurr .set     A13      ; Current delay state pointer.
325 TT0     .set     A1       ; Temporary for testing purpose.
326 TT1     .set     B1       ; Temporary for testing purpose.
327 Q0      .set     A3       ; Temporary with a handy name.
328 Q1      .set     B3       ; Temporary with a handy name.
329 W0      .set     A6       ; Temporary with a handy name.
330 W1      .set     B6       ; Temporary with a handy name.

```



```
331
332 *** Registers name that will be used for in Step 3 and Step 4.
333 DS0      .set    A0      ; Delay state 0
334 DS1      .set    B0      ; Delay state 1
335 DS2      .set    A1      ; Delay state 2
336 DS3      .set    B1      ; Delay state 3
337 DS4      .set    A2      ; Delay state 4
338 DS5      .set    B2      ; Delay state 5
339 DS6      .set    A3      ; Delay state 6
340 DS7      .set    B3      ; Delay state 7
341 Dist0     .set    A7      ; Distance 0
342 Dist1     .set    B7      ; Distance 1
343
344 *** Registers name that will be used for in Step 4 and Step 5.
345 G0        .set    A5      ; Accumulate distance 0.
346 G1        .set    B5      ; Accumulate distance 1.
347 G2        .set    A10     ; Accumulate distance 2.
348 G3        .set    B10     ; Accumulate distance 3.
349 G4        .set    A11     ; Accumulate distance 4.
350 G5        .set    B11     ; Accumulate distance 5.
351 G6        .set    A9      ; Accumulate distance 6.
352 G7        .set    B9      ; Accumulate distance 7.
353
354 *** Registers name that will be used for in Step 5 and Step 6.
355 TTT       .set    A8      ; Traced back path state result.
356
357 *** Program start label.
358         .def      _Viterbi
359         .text
360 _Viterbi:
361
362 *** Step 1 - Opening statements.          ;*
363 *** Purpose - Allocate stack to store B3, A10-A13, and B10-B13.
364 *** Place argument 3 (A6) to MCurr (bss).
365 *** Setup current delay state pointer to DStCurr.
366
367         MV        A6,B6          ;*
368 ||      LDW        *A6,MCurr0     ;** Get first modulation table.
369 ||      SUBAW      B15,9,B15      ;* Allocate storage.
370
371         MVK        MCurr,B0       ;*
372 ||      MVK        DStCurrbss,A1 ;*
373 ||      MV         B15,A9         ;* Set up a A register save faster.
374 ||      STW        B3,*+B15[1]    ;* Save registers.
375
376         MVKH       MCurr,B0       ;*
377 ||      MVKH       DStCurrbss,A1 ;*
378 ||      STW        A13,*+A9[8]    ;* Save registers.
379 ||      STW        B13,*+B15[9]   ;* Save registers.
380
381         STW        B6,*B0         ;* Save current pointer first.
382 ||      LDW        *A1,DStCurr    ;* Get Delay state pointer.
383
384         STW        A12,*+A9[6]    ;* Save registers.
385 ||      STW        B12,*+B15[7]   ;* Save registers.
386
387 *** Step 2 - Distances and states.        ;**
388 *** Purpose - Calculate the Distance and State value
389 *** based on the input Xi and Yi.
390 *** Input: Xi, Yi (registers)
391 *** Output: Distance, State (bss)
392
393 XY       .set    A0      ; X and Y for SUB2 operation.
394 Cc       .set    B0      ; Loop counter.
395 Xt       .set    A1      ; Test X
396 Yt       .set    B1      ; Test Y
397 S1       .set    A1      ; State 1
```



```

398 S0      .set    B1      ; State 0
399 TT      .set    B2      ; Temporary
400 Nx      .set    A3      ; Nx
401 Ny      .set    B3      ; Ny
402 Xi      .set    A4      ; Symbol received - Argument 1
403 Yi      .set    B4      ; Symbol received - Argument 2
404 Xs      .set    A5      ; Xs
405 Ys      .set    B5      ; Ys
406 MCurr0  .set    A6      ; Bit rate modulation table to use
407 StateP  .set    B6      ; Points to state result
408 Xq      .set    A7      ; Xq
409 Yq      .set    B7      ; Yq
410 DistP   .set    A8      ; Points to distance result
411 MCurr1  .set    B8      ; Table pointer for B register file
412 Ix      .set    A9      ; X index
413 Iy      .set    B9      ; Y index
414 Id0     .set    A9      ; Region result
415 Id1     .set    B9      ; Region result for B register file
416 P2      .set    A9      ; Product register
417 P3      .set    B9      ; Product register
418 T0      .set    A10     ; Temporary register
419 T1      .set    B10     ; Temporary register
420 P0      .set    A10     ; Product register
421 P1      .set    B10     ; Product register
422 T2      .set    A11     ; Temporary register
423 T3      .set    B11     ; Temporary register
424 MNext0  .set    A12     ; Next table pointer
425
426 Step2:
427         MV      MCurr0,TT      ;**
428         ||      MVK      State,StateP      ;** Set up State result pointer.
429         ||      MVK      Distance,DistP     ;** Set up Distance result pointer.
430         ||      STW      A11,*+A9[4]        ;* Save registers.
431         ||      STW      B11,*+B15[5]       ;* Save registers.
432
433         LDH      *+MCurr0[3],Xs      ;** Get Xs and Ys from table.
434         ||      LDH      *+TT[2],Ys      ;**
435         ||      MVKH     State,StateP     ;**
436         ||      MVKH     Distance,DistP    ;**
437
438         LDH      *+MCurr0[5],Xq      ;** Get Xq and Yq from table.
439         ||      LDH      *+TT[4],Yq      ;**
440
441         LDH      *+MCurr0[7],Nx      ;** Get Nx and Ny from table.
442         ||      LDH      *+TT[6],Ny      ;**
443         ||      ZERO     Ix           ;** Initialize indexes.
444         ||      ZERO     Iy           ;**
445         ||      MVK      8,Cc         ;** Initialize loop counter.
446
447         STW      A10,*+A9[2]         ;* Save registers.
448         ||      STW      B10,*+B15[3]    ;* Save registers.
449
450         NOP      2                   ;**
451
452 Cloop:
453         CMPGT    Xi,Xs,Xt            ;** Compare and increment test level.
454         ||      ADD      Xs,Xq,Xs      ;**
455         ||      CMPGT    Yi,Ys,Yt      ;**
456         ||      ADD      Ys,Yq,Ys      ;**
457         ||      LDW      *MCurr0,MNext0 ;**^ Prepare for next iteration.
458         ||      MV      MCurr0,TT      ;**
459
460         .loop     4                   ;** Set to Max (Nx, Ny).
461         CMPGT    Xi,Xs,Xt            ;** Compare and increment test level.
462         || [Xt]   ADD      Xs,Xq,Xs      ;**
463         || [Xt]   ADD      Ix,3,Ix      ;** Also increase indexes.
464         ||      CMPGT    Yi,Ys,Yt      ;**

```



```

465      || [Yt]   ADD     Ys,Yq,Ys           ;**
466      || [Yt]   ADD     Iy,Nx,Iy           ;**
467      ||         .endloop                   ;**
468
469      ||         ADD     Ix,Iy,Id0           ;** Calculate region.
470      ||         ADD     Ix,Iy,Id1           ;**
471      ||         ADDAH   MCurr0,8,MCurr0     ;** Points to path state start.
472      ||         ADDAH   TT,10,MCurr1        ;** Pointer for B register file.
473
474      ||         LDW     *MCurr0[Id0],T0      ;** Get X0, Y0.
475      ||         LDW     *MCurr1[Id1],T1      ;** Get X1, Y1.
476      ||         ADD     Id0,2,Id0           ;**
477
478      ||         LDW     *MCurr0[Id0],T2      ;** Get State 0 and 1.
479
480      ||         SHL     Xi,16,XY            ;** Combine Xi Yi to XY.
481      ||         CLR     Yi,16,31,TT         ;**
482      ||         LDH     *+MNext0[2],Ys       ;**^ Prepare for next iteration.
483
484      ||         OR      XY,TT,XY            ;** Combine Xi Yi to XY.
485      ||         LDH     *+MNext0[3],Xs       ;**^ Prepare for next iteration.
486
487      ||         LDH     *+MNext0[4],Yq       ;**^ Prepare for next iteration.
488
489      ||         SUB2    T0,XY,T0            ;** Get X Y difference from X0 Y0.
490      ||         SUB2    T1,XY,T1            ;** Get X Y difference from X1 Y1.
491      ||         LDH     *+MNext0[5],Xq       ;**^ Prepare for next iteration.
492
493      ||         MPY     T0,T0,P2            ;** Get (Yi - Y0) ^ 2.
494      ||         MPY     T1,T1,P3            ;** Get (Yi - Y1) ^ 2.
495      ||         LDH     *+MNext0[6],Ny       ;**^ Prepare for next iteration.
496      || [Cc]   SUB     Cc,1,Cc              ;** Decrement loop counter.
497
498      ||         MPYH    T0,T0,P0            ;** Get (Xi - X0) ^ 2.
499      ||         MPYH    T1,T1,P1            ;** Get (Yi - Y0) ^ 2.
500      ||         LDH     *+MNext0[7],Nx       ;**^ Prepare for next iteration.
501      || [Cc]   B       Cloop                ;** Next iteration.
502
503      ||         NOP                                     ;** Delay slot for multiply.
504
505      ||         ADD     P0,P2,T0            ;** Get distance square from X0 Y0.
506      ||         ADD     P1,P3,T1            ;** Get distance square from X1 Y1.
507      ||         MV      T2,T3              ;** State 0 and 1 info here.
508
509      ||         CMPGT   T0,T1,TT            ;** Which distance is larger.
510      ||         EXTU    T3,16,16,S0         ;** Separate state 0 and 1.
511      ||         EXTU    T2,0,16,S1         ;**
512
513      || [TT]   MV      T1,T0                ;** If distance from X0 Y0 larger.
514      || [TT]   MV      S1,S0                ;** then use distance X1 Y1.
515
516      ||         STW     T0,*DistP++         ;** Save results.
517      ||         STH     S0,*StateP++        ;**
518      ||         MV      MNext0,MCurr0       ;**^ Prepare for next iteration.
519      ||         ZERO    Ix                  ;**^ Reset indexes.
520      ||         ZERO    Iy                  ;**^
521      ||         ; Branch to Cloop here.
522
523 *** Step 3 - Shortest distance           ;**
524 *** Purpose - Find the shortest distances and place it in Dist0
525 *** and Dist1. From the distance indexes, get the delay
526 *** states and store them.
527 *** Input: Distance, State (bss)
528 *** Output: Dist0, Dist1 (registers)
529 *** DS0 - DS7 (registers)
530 *** DelayPath (bss)
531

```



```

532 TMB0      .set    A0      ; Table pointer
533 TMB1      .set    B0      ; Table pointer
534 T200A     .set    A2      ; Store 200 hex
535 T200B     .set    B2      ; Store 200 hex
536 DistP0    .set    A4      ; Distance pointer
537 DistP1    .set    B4      ; Distance pointer
538 Ix0       .set    A5      ; Index
539 Ix1       .set    B5      ; Index
540 DSP0      .set    A6      ; Delay state pointer
541 DSP1      .set    B6      ; Delay state pointer
542 Stp0      .set    A8      ; State pointer 0
543 Stp1      .set    B8      ; State pointer 1
544 St0       .set    A9      ; State 0
545 St1       .set    B9      ; State 1
546 LU0       .set    A10     ; Lookup table row 0
547 LU1       .set    B10     ; Lookup table row 1
548 LU2       .set    A11     ; Lookup table row 2
549 LU3       .set    B11     ; Lookup table row 3
550
551 Step3:
552         MVK      Distance,DistP0      ;*** Points to Distance 0 to 3.
553         ||       MVK      Distance,DistP1      ;*** Points to Distance 4 to 7.
554
555         MVKH     Distance,DistP0      ;***
556         ||       MVKH     Distance,DistP1      ;***
557
558         LDW      *+DistP0[0],Dist0     ;*** Get Distance 0.
559         ||       LDW      *+DistP1[4],Dist1   ;*** Get Distance 4.
560         ||       MVK      TMBBack,TMB0      ;*** Get table pointer.
561         ||       MVK      TMBBack,TMB1      ;*** Get table pointer.
562
563         LDW      *+DistP0[1],Q0        ;*** Get Distance 1.
564         ||       LDW      *+DistP1[5],Q1        ;*** Get Distance 5.
565         ||       MVKH     TMBBack,TMB0      ;***
566         ||       MVKH     TMBBack,TMB1      ;***
567
568         LDW      *+TMB0[0],LU0         ;*** Read lookup table.
569         ||       LDW      *+TMB1[1],LU1         ;*** Read lookup table.
570         ||       MVK      State,Stp0        ;*** Get state pointer.
571         ||       MVK      State,Stp1        ;*** Get state pointer.
572
573         LDW      *+DistP0[2],Q0        ;*** Get Distance 2.
574         ||       LDW      *+DistP1[6],Q1        ;*** Get Distance 6.
575         ||       MVKH     State,Stp0        ;*** Get state pointer.
576         ||       MVKH     State,Stp1        ;*** Get state pointer.
577
578         LDW      *+TMB0[2],LU2         ;*** Read lookup table.
579         ||       LDW      *+TMB1[3],LU3         ;*** Read lookup table.
580
581         MVK      0x200,T200A          ;***
582         ||       MVK      0x200,T200B          ;***
583         ||       LDW      *+DistP0[3],Q0        ;*** Get Distance 3.
584         ||       LDW      *+DistP1[7],Q1        ;*** Get Distance 7.
585
586         CMPGT    Dist0,Q0,TT0          ;*** If Distance 1 < lowest.
587         ||       MVK      0x39C,Ix0          ;*** Extract bit 0 - 3.
588         ||       CMPGT    Dist1,Q1,TT1        ;*** If Distance 5 < lowest.
589         ||       MVK      0x39C,Ix1          ;*** Extract bit 0 - 3.
590         ||       LDH      *Stp0[0],St0        ;***
591         ||       LDH      *Stp1[4],St1        ;***
592
593         [TT0]    MV      Q0,Dist0          ;*** Keep lowest distances.
594         ||       [TT0]    MVK      0x31C,Ix0          ;*** Extract bit 4 - 7.
595         ||       [TT0]    LDH      *Stp0[1],St0        ;***
596         ||       [TT1]    MV      Q1,Dist1        ;*** Keep lowest distances.
597         ||       [TT1]    MVK      0x31C,Ix1          ;*** Extract bit 4 - 7.
598         ||       [TT1]    LDH      *Stp1[5],St1        ;***

```



```

599
600          CMPGT    Dist0,Q0,TT0          ;*** If Distance 2 < lowest.
601      ||          CMPGT    Dist1,Q1,TT1          ;*** If Distance 6 < lowest.
602
603      [TT0]  MV      Q0,Dist0          ;*** Keep lowest distances.
604      || [TT0]  MVK      0x29C,Ix0          ;*** Extract bit 8 - 11.
605      || [TT0]  LDH      *Stp0[2],St0          ;***
606      || [TT1]  MV      Q1,Dist1          ;*** Keep lowest distances.
607      || [TT1]  MVK      0x29C,Ix1          ;*** Extract bit 8 - 11.
608      || [TT1]  LDH      *Stp1[6],St1          ;***
609
610          CMPGT    Dist0,Q0,TT0          ;*** If Distance 3 < lowest.
611      ||          CMPGT    Dist1,Q1,TT1          ;*** If Distance 7 < lowest.
612      ||          MV      DStCurr,DSP0          ;*** Initialize pointer for A side.
613      ||          MV      DStCurr,DSP1          ;*** Initialize pointer for B side.
614
615      [TT0]  MV      Q0,Dist0          ;*** Keep lowest distances.
616      || [TT0]  MVK      0x21C,Ix0          ;*** Extract bit 12 - 15.
617      || [TT0]  LDH      *Stp0[3],St0          ;***
618      || [TT1]  MV      Q1,Dist1          ;*** Keep lowest distances.
619      || [TT1]  MVK      0x21C,Ix1          ;*** Extract bit 12 - 15.
620      || [TT1]  LDH      *Stp1[7],St1          ;***
621
622          EXTU      LU0,Ix0,DS0          ;*** Do the extraction.
623      ||          EXTU      LU1,Ix1,DS1          ;*** Do the extraction.
624
625          EXTU      LU2,Ix0,DS2          ;*** Do the extraction.
626      ||          EXTU      LU3,Ix1,DS3          ;*** Do the extraction.
627      ||          SUB      Ix0,T200A,Ix0          ;*** Extract range is upper 16 bit.
628      ||          SUB      Ix1,T200B,Ix1          ;*** Extract range is upper 16 bit.
629      ||          STH      DS0,*+DSP0[0]          ;*** Save to DelayState 0.
630      ||          STH      DS1,*+DSP1[1]          ;*** Save to DelayState 1.
631
632          EXTU      LU0,Ix0,DS4          ;*** Do the extraction.
633      ||          EXTU      LU1,Ix1,DS5          ;*** Do the extraction.
634      ||          STH      DS2,*+DSP0[2]          ;*** Save to DelayState 2.
635      ||          STH      DS3,*+DSP1[3]          ;*** Save to DelayState 3.
636
637          EXTU      LU2,Ix0,DS6          ;*** Do the extraction.
638      ||          EXTU      LU3,Ix1,DS7          ;*** Do the extraction.
639      ||          STH      DS4,*+DSP0[4]          ;*** Save to DelayState 4.
640      ||          STH      DS5,*+DSP1[5]          ;*** Save to DelayState 5.
641
642          STH      DS6,*+DSP0[6]          ;*** Save to DelayState 6.
643      ||          STH      DS7,*+DSP1[7]          ;*** Save to DelayState 7.
644      ||          MVK      AccDist,AccP0          ;****
645      ||          MVK      AccDist,AccP1          ;****
646
647          STH      St0,*+DSP0[8]          ;*** Save Path-state info.
648      ||          STH      St1,*+DSP1[9]          ;*** Save Path-state info.
649      ||          MVKH      AccDist,AccP0          ;****
650      ||          MVKH      AccDist,AccP1          ;****
651
652      *** Step 4 - Accumulate distance          ;****
653      *** Purpose - Calculate the accumulated distances using a fix formula.
654      *** Input: DS0 - DS7 (registers)
655      *** Dist0, Dist1 (registers)
656      *** Output: AccDist (bss)
657      *** G0 - G7 (registers)
658
659      AccP0 .set      A4          ; Accumulate Distance pointer
660      AccP1 .set      B4          ; Accumulate Distance pointer
661
662      Step4:
663          LDW      *+AccP0[DS0],G0          ;**** Read accumulate distance.
664      ||          LDW      *+AccP1[DS1],G1          ;****
665

```




```

666          LDW      *+AccP0[DS2],G2      ;**** Read accumulate distance.
667      ||          LDW      *+AccP1[DS3],G3      ;****
668
669          LDW      *+AccP0[DS4],G4      ;**** Read accumulate distance.
670      ||          LDW      *+AccP1[DS5],G5      ;****
671
672          LDW      *+AccP0[DS6],G6      ;**** Read accumulate distance.
673      ||          LDW      *+AccP1[DS7],G7      ;****
674      ||          MVK      1,ONE          ;***** ONE = 1.
675
676          SHR      Dist0,3,Dist0        ;**** Divide Distances by 8.
677      ||          SHR      Dist1,3,Dist1        ;****
678
679          SHR      G0,3,W0              ;**** Divide accumulate distances by 8.
680      ||          SHR      G1,3,W1              ;****
681
682          SUB      G0,W0,G0              ;**** Subtract to get 7/8 of itself.
683      ||          SUB      G1,W1,G1              ;****
684      ||          SHR      G2,3,W0              ;***** Divide accumulate distance by 8.
685      ||          SHR      G3,3,W1              ;*****
686      ||          MPY      ONE,0,Q0          ;***** Q0 index at 0.
687      ||          MPY      ONE,1,Q1          ;***** Q1 index at 1.
688
689          ADD      G0,Dist0,G0          ;**** Add the distances.
690      ||          ADD      G1,Dist1,G1          ;****
691      ||          SUB      G2,W0,G2              ;***** Subtract to get 7/8 of itself.
692      ||          SUB      G3,W1,G3              ;*****
693      ||          SHR      G4,3,W0              ;***** Divide accumulate distance by 8.
694      ||          SHR      G5,3,W1              ;*****
695
696          STW      G0,*+AccP0[0]        ;**** Store accumulate distances.
697      ||          STW      G1,*+AccP1[1]        ;****
698      ||          ADD      G2,Dist0,G2          ;***** Add the distances.
699      ||          ADD      G3,Dist1,G3          ;*****
700      ||          SUB      G4,W0,G4              ;***** Subtract to get 7/8 of itself.
701      ||          SUB      G5,W1,G5              ;*****
702
703          STW      G2,*+AccP0[2]        ;***** Store accumulate distances.
704      ||          STW      G3,*+AccP1[3]        ;*****
705      ||          ADD      G4,Dist0,G4          ;***** Add the distances.
706      ||          ADD      G5,Dist1,G5          ;*****
707      ||          SHR      G6,3,W0              ;***** Divide accumulate distance by 8.
708      ||          SHR      G7,3,W1              ;*****
709
710          STW      G4,*+AccP0[4]        ;***** Store accumulate distances.
711      ||          STW      G5,*+AccP1[5]        ;*****
712      ||          SUB      G6,W0,G6              ;***** Subtract to get 7/8 of itself.
713      ||          SUB      G7,W1,G7              ;*****
714      ||          CMPGT    G0,G2,TT0          ;***** Find out which is smaller.
715      ||          CMPGT    G1,G3,TT1          ;*****
716
717          ADD      G6,Dist0,G6          ;***** Add the distances.
718      ||          ADD      G7,Dist1,G7          ;*****
719      ||      [TT0]    MV      G2,G0              ;***** Save smaller.
720      ||      [TT0]    MPY      ONE,2,Q0          ;***** Update index.
721      ||      [TT1]    MV      G3,G1              ;***** Save smaller.
722      ||      [TT1]    MPY      ONE,3,Q1          ;***** Update index.
723
724          STW      G6,*+AccP0[6]        ;***** Store accumulate distances.
725      ||          STW      G7,*+AccP1[7]        ;*****
726      ||          CMPGT    G0,G1,TT0          ;***** Find out which is smaller.
727      ||          MPY      ONE,4,W0              ;***** W0 index at 4.
728      ||          MPY      ONE,5,W1              ;***** W1 index at 5.
729
730      *** Step 5 - Backward trace          ;*****
731      *** Purpose - First find the smallest accumulated distance, then
732      *** trace backwrdr using DelayPath to find the path-state.

```



```
733 ***                      Increment the delay state pointer DStCurrbss.
734 ***                      Input: G0 - G7 (registers)
735 ***                      DelayPath (bss)
736 ***                      Output: TTT (registers)
737
738 II      .set      B0      ; Temporary.
739 IDD     .set      A4      ; Index.
740 DStptr  .set      A5      ; Delay State pointer.
741 JJ      .set      A7      ; Temporary.
742 JJ0     .set      B8      ; Temporary.
743 ONE     .set      A12     ; One
744
745 Step5:
746     [TT0] MV      G1,G0      ;***** Minimum of first four found.
747     || [!TT0] MV      Q0,Q1      ;***** Value in G0, index in Q1.
748     ||      CMPGT   G4,G6,TT0      ;***** Find minimum of next four.
749     ||      CMPGT   G5,G7,TT1      ;*****
750
751     [TT0] MV      G6,G4      ;***** Save smaller.
752     || [TT0] MPY     ONE,6,W0      ;***** Update index.
753     || [TT1] MV      G7,G5      ;***** Save smaller.
754     || [TT1] MPY     ONE,7,W1      ;***** Update index.
755     ||      MVK     MCurr,MCurrx      ;***** Get back pointer.
756
757     CMPGT   G4,G5,TT0      ;***** Find out which is smaller.
758     ||      MVKH    MCurr,MCurrx      ;*****
759
760     [TT0] MV      G5,G4      ;***** Minimum of last four found.
761     || [!TT0] MV      W0,W1      ;***** Value in G4, index in W1.
762     ||      MVK     DelayPath,JJ0      ;***** Prepare for circular buffer for JJ.
763
764     CMPGT   G0,G4,TT0      ;***** Get to find the real minimum.
765     ||      MV      Q1,IDD      ;***** Place index.
766     ||      MVKH    DelayPath,JJ0      ;***** Prepare for circular buffer for JJ.
767
768     [TT0] MV      W1,IDD      ;***** Got index at IDD.
769     ||      MV      DStCurr,JJ      ;***** Get JJ for trace back.
770     ||      MVK     14,II      ;***** Trace back counter.
771     ||      LDW     *MCurrx,MCurrx      ;***** Load the pointer.
772
773 IILoop:
774     [II] B      IILoop      ;***** Trace back.
775
776     LDH      *+JJ[IDD],IDD      ;***** Keep tracing.
777     ||      MVK     Diff,DFTP      ;***** Get diff table pointer.
778
779     CMPGT   JJ,JJ0,TT0      ;***** Do JJ circular buffer.
780     ||      MVKH    Diff,DFTP      ;***** Get diff table pointer.
781
782     [TT0] SUBAH   JJ,10,JJ      ;***** Adjust JJ in circular fashion.
783     || [!TT0] MVK     DelayPath+300,JJ ;***** Adjust JJ in circular fashion.
784
785     [!TT0] MVKH    DelayPath+300,JJ ;***** Adjust JJ in circular fashion.
786
787     SUB      II,1,II      ;***** Decrement counter.
788     ; Branch to IILoop here.
789
790     AND      IDD,1,IDD      ;***** LSB decides which path-state to pick.
791     ||      LDW     *+MCurrx[1],Ext1 ;***** Get extraction indexes.
792     ||      MVK     Tdr,TDRP      ;***** Get Tdr pointer.
793
794     ADD      8,IDD,IDD      ;***** Index it to path-state in DelayPath.
795     ||      LDW     *+MCurrx[2],Ext2 ;***** Get extraction indexes.
796     ||      MVKH    Tdr,TDRP      ;***** Get Tdr pointer.
797
798     LDH      *+JJ[IDD],TTT      ;***** Get the path-state.
799     ||      MVK     DelayPath+300,JJ0 ;***** Prepare for cir buffer for DStCurr.
```



```

800      ||          MVK      DStCurrbss,DStptr      ;***** Get DStCurr pointer.
801
802          MVKH      DelayPath+300,JJ0      ;***** Prepare for cir buffer for DStCurr.
803      ||          MVKH      DStCurrbss,DStptr      ;***** Get DStCurr pointer.
804      ||          LDW      *TDRP,TDR      ;***** Read Tdr.
805
806          CMPLT      DStCurr,JJ0,TT0      ;***** Do DStCurr circular buffer.
807      ||          LDW      *+MCurrx[3],Sht      ;***** Get shift indexes.
808
809          [TT0]      ADDAH      DStCurr,10,DStCurr      ;***** Adjust DStCurr in circular fashion.
810      ||      [!TT0]      MVK      DelayPath,DStCurr      ;***** Adjust DStCurr in circular fashion.
811      ||          LDW      *+B15[1],B3      ;***** Return sequences.
812
813          [!TT0]      MVKH      DelayPath,DStCurr      ;***** Adjust DStCurr in circular fashion.
814      ||          LDW      *DFTP,DFT      ;***** Read diff table.
815
816      *** Step 6 - Differentiate      ;*****
817      ***      Purpose - Differentiate the result.
818      ***      Input: TTT (registers)
819      ***      Output: A4 (registers) return value.
820
821      MCurrx      .set      A0      ; Modulation table pointer.
822      PP      .set      A0      ; Temporary.
823      Ext2      .set      A2      ; Extraction index 2.
824      TDR      .set      B2      ; Tdr value.
825      SSS      .set      A3      ; Temporary.
826      Sht      .set      A4      ; Shift amount.
827      TDRP      .set      B4      ; Tdr pointer.
828      Ext1      .set      A6      ; Extraction index 1.
829      DFT      .set      A7      ; Diff table value.
830      DFTP      .set      A9      ; Diff table pointer.
831
832      Step6:
833          STW      DStCurr,*DStptr      ;***** Save DStCurr.
834      ||          EXTU      TTT,Ext1,PP      ;***** Separate TTT into two pieces.
835
836          SHL      PP,2,PP      ;***** Get diff table index.
837      ||          STW      PP,*TDRP      ;***** Update new Tdr value.
838
839          OR      PP,TDR,PP      ;***** Get diff table index.
840      ||          EXTU      TTT,Ext2,SSS      ;***** Separate TTT into two pieces.
841
842          SHL      PP,1,PP      ;***** Get diff table index.
843      ||          MV      B15,A9      ;***** Return sequences.
844      ||          B      B3      ;***** Return sequences.
845
846          SHR      DFT,PP,DFT      ;***** Differentiate.
847      ||          LDW      *+A9[2],A10      ;***** Return sequences.
848      ||          LDW      *+B15[3],B10      ;***** Return sequences.
849
850          CLR      DFT,2,31,DFT      ;***** Keep lower 2 bits.
851      ||          LDW      *+A9[4],A11      ;***** Return sequences.
852      ||          LDW      *+B15[5],B11      ;***** Return sequences.
853
854          SHL      DFT,Sht,DFT      ;***** Combine with SSS.
855      ||          LDW      *+A9[6],A12      ;***** Return sequences.
856      ||          LDW      *+B15[7],B12      ;***** Return sequences.
857
858          OR      DFT,SSS,A4      ;***** Combine with SSS.
859      ||          LDW      *+A9[8],A13      ;***** Return sequences.
860      ||          LDW      *+B15[9],B13      ;***** Return sequences.
861
862          ADDAW      B15,9,B15      ;***** Deallocate storage.
863          ; Branch back to calling routine here.
864
865      *** Step 7 - Return      ;*****
866      ***      Purpose - Restore registers A10 - A13, B10 - B13, and A3.

```



```
867 ***                               Jump back to calling routine.
868 Step7:
869         .end
870
```



Appendix C. Main Program to Test the Algorithm

```

1  /*****
2  * File:      main.c
3  * Purpose:   main C program to test the performance of the Viterbi code.
4  *****/
5
6  #include "main.h"
7
8  /*****
9  * Trellis - encoder.
10 *****/
11
12 /* Delay states transition matrix looking forward from present state. */
13 char TMForward[8][4] = {
14     0, 6, 2, 4,
15     2, 4, 0, 6,
16     4, 2, 6, 0,
17     6, 0, 4, 2,
18     1, 5, 7, 3,
19     3, 7, 5, 1,
20     7, 3, 1, 5,
21     5, 1, 3, 7
22 };
23
24 extern Mod M14C;
25 Mod *Mcurr = &M14C;
26 char Sht = 4;
27 char Tdt = 0, TransmitState = 0;
28
29 void Trellis (int DataIn, int *Xo, int *Yo)
30 {
31     /* Differentiator truth table. Bit 4,5 for transmitter, bit 0,1 receiver. */
32     char Diff[] = { 0x00, 0x11, 0x23, 0x32, 0x11, 0x00, 0x32, 0x23,
33                    0x22, 0x33, 0x10, 0x01, 0x33, 0x22, 0x01, 0x10 };
34     short tt, ss, pp;
35     tt = (DataIn >> Sht) & 0x03;
36     tt = (tt << 2) | Tdt;
37     tt = (Diff[tt] >> 4) & 0x03;
38     Tdt = tt;
39     pp = tt | (TransmitState & 0x04);
40     TransmitState = TMForward[TransmitState][tt];
41     ss = DataIn & ((1 << Sht) - 1);
42     ss = *(Mcurr->Pt[pp] + ss);
43     *Xo = *(Mcurr->Xc[pp] + ss) << 10;
44     *Yo = *(Mcurr->Yc[pp] + ss) << 10;
45 }
46
47 /*****
48 * ReadData - setup test data.
49 *****/
50
51 int ReadData ()
52 {
53     int DataIn;
54     static short i = 0;
55     static short TestData[] = {
56         0x01, 0x11, 0x31, 0x01, 0x11, 0x31, 0x01, 0x11, 0x31, 0x00
57     };
58     DataIn = TestData[i];
59     i++; if (i >= 10) i = 0;
60     return (DataIn);
61 }
62
63 /*****
64 * OutData - output test data.

```



```
65 *****/
66
67 void OutData (int DataOut)
68 {
69     /* ?? = DataOut; */
70 }
71
72 /*****
73 * main - Run test.
74 *****/
75
76 extern int M14;
77 #define Mod14 &M14
78
79 int Viterbi (int, int, int*);
80 int Init (void);
81
82 main ()
83 {
84     int count = 0;
85     int DataIn, DataOut, Xo, Yo, Xi, Yi;
86     Init ();
87     for (;;)
88     {
89         DataIn = ReadData();
90         Trellis (DataIn, &Xo, &Yo);
91         Xi = Xo; Yi = Yo;
92 #ifdef AA
93         DataOut = Viterbi (Xi,Yi,Mod14);          /* For assembly Viterbi */
94 #endif
95 #ifdef CC
96         DataOut = Viterbi (Xi,Yi,(int*)&M14C);    /* For C Viterbi */
97 #endif
98         OutData (DataOut);
99         count++;
100     }
101 }
102
```