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

TEXAS INSTRUMENTS

TRADEMARKS

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

Figures

Tables

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

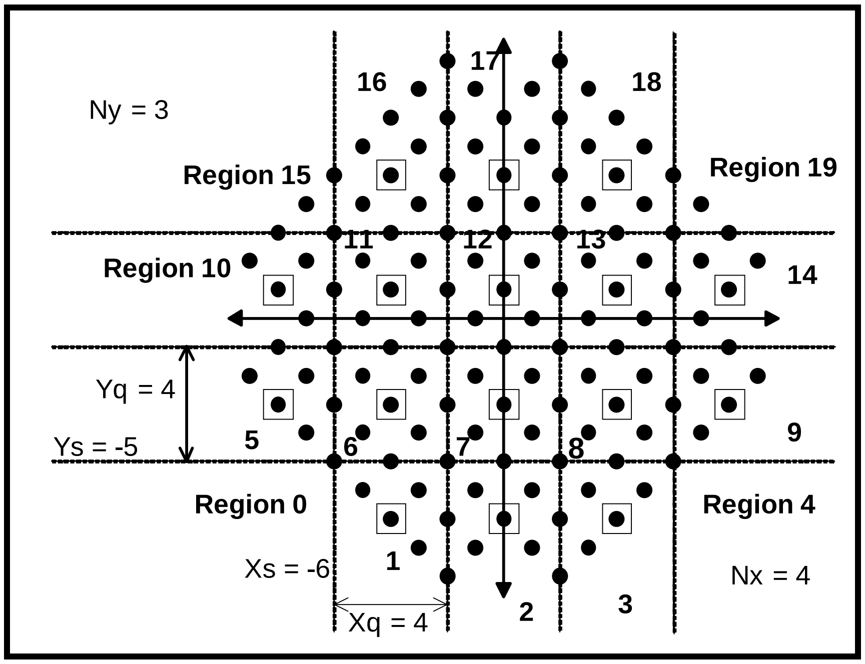


Figure 2. Dividing the Signal Space Diagram into Regions for 12 kbps – with Path-Index (Y0, Y1, and Y2) = 000

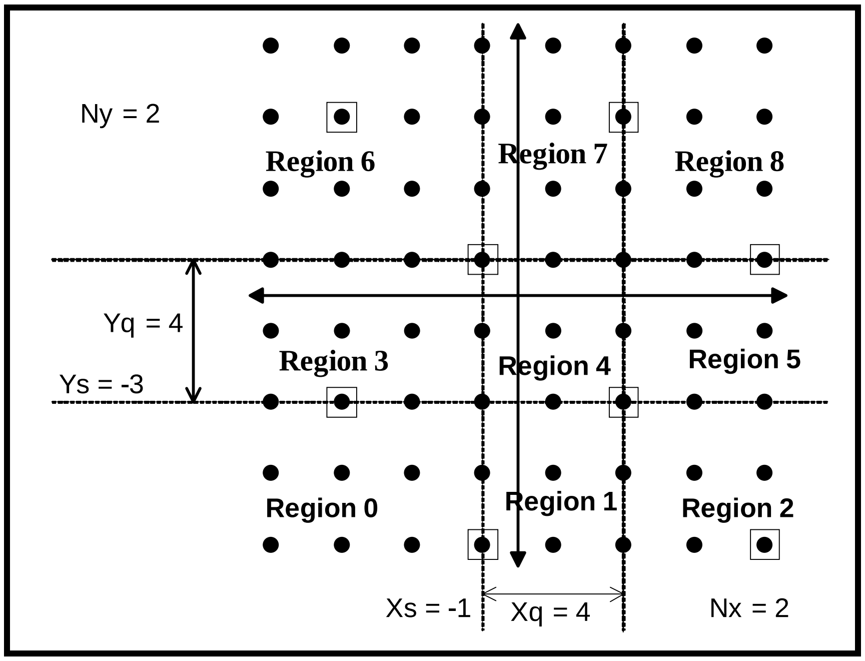


Figure 3. Dividing the Signal Space Diagram into Regions for 9.6 kbps – with Path-Index (Y0, Y1, and Y2) = 000

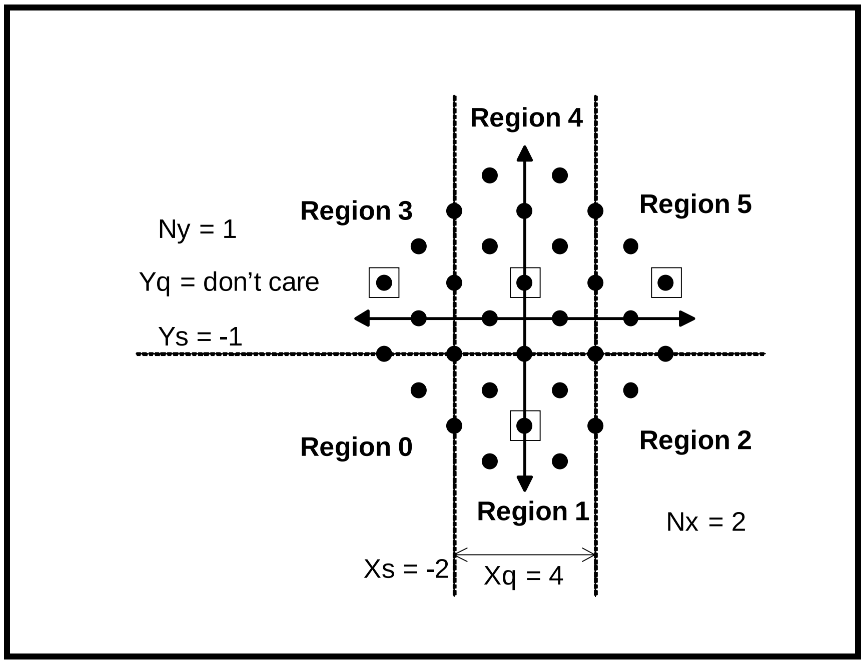
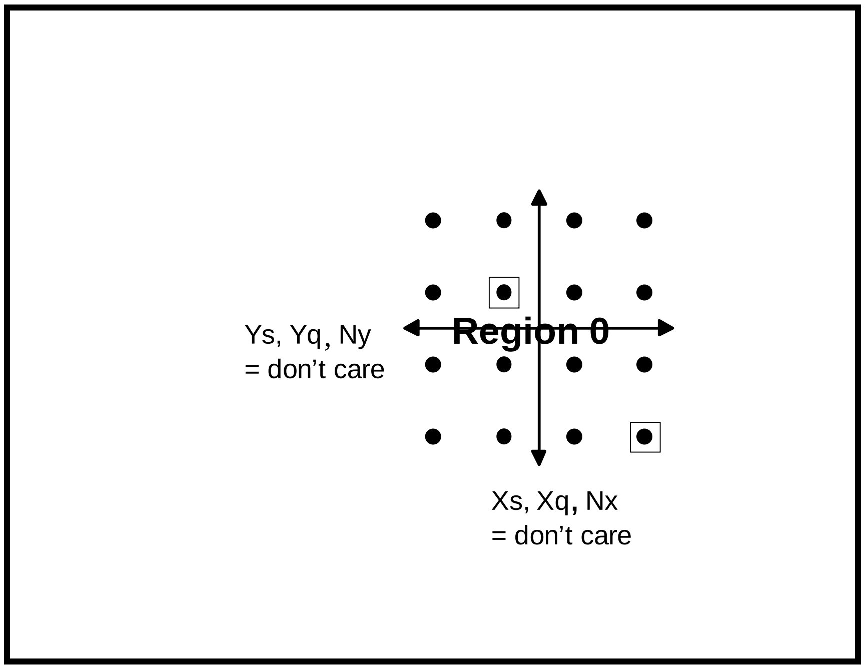


Figure 4. Dividing the Signal Space Diagram into Regions for 7.2 kbps – with Path-Index (Y0, Y1, and Y2) = 000



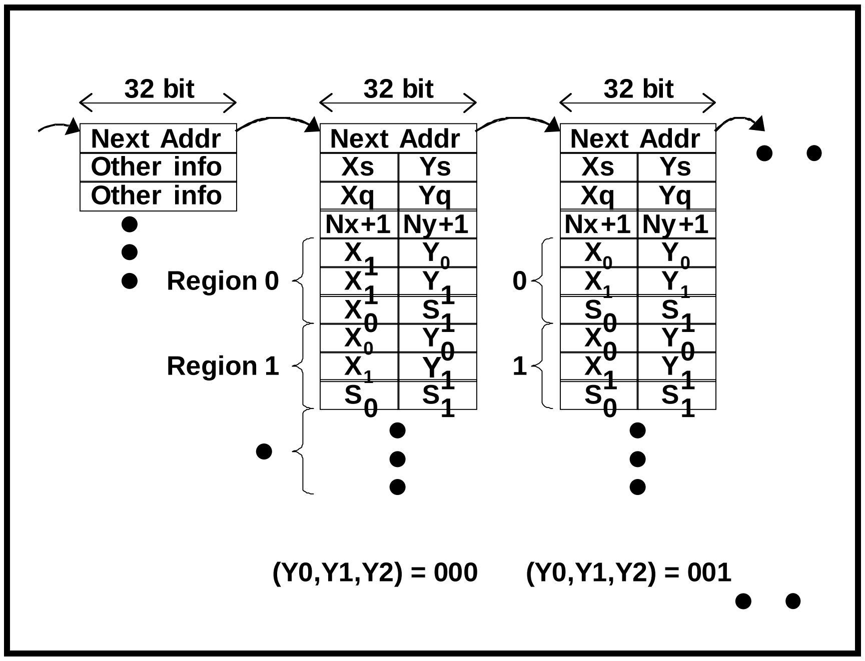
The diagrams show that once the region of the received symbol is determined, it is only necessary to calculate the distances from at most two path-states to find out which one is closest.

In the figures, Xs and Ys represent the lowermost and leftmost X and Y locations of the dividing line. Xq and Yq represent the X and Y distances between the dividing lines. Nx and Ny 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 (Y0, Y1, and Y2). 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 (Y0, Y1, and Y2) 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 (Y0,Y1,Y2) = 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 (Y0,Y1,Y2) = 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 (Y0,Y1,Y2) = 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 (Y0,Y1,Y2) = 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.

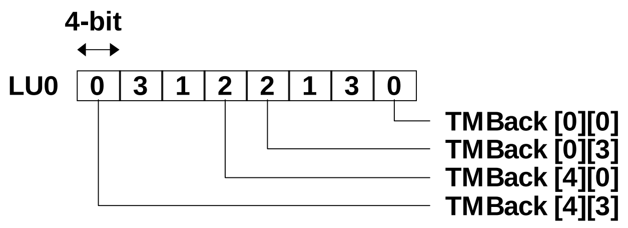
The outputs from this step are the previous delay-states stored in the delay-state registers DS0 through DS7. These will be used immediately by the next step. The same values are being stored in the DelayPath .bss section, pointed to by DStCurr. The data structure of the DelayPath array is shown in Figure 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 TMBack Look-up Table



```
EXTU          LU0, Ix0, DS0
; if Ix0 = 0x39C, get   TMBack [0][0]
; if Ix0 = 0x19C, get     “   “   [4][0]
; if Ix0 = 0x31C, get     “   “   [0][1]
; if Ix0 = 0x11C, get     “   “   [4][1]
; if Ix0 = 0x29C, get     “   “   [0][2]
; if Ix0 = 0x09C, get     “   “   [4][2]
; if Ix0 = 0x21C, get     “   “   [0][3]
; if Ix0 = 0x01C, get     “   “   [4][3]
```

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 Ix0 for the EXTU instruction.Thus, we can use Ix0 to extract TMBack[0][0]. Next, by setting bit 9 of Ix0 to zero (this is done by a SUB 0x200 instruction), we can extract TMBack[4][0].

Step 4. Calculate Accumulate Distances

C Code:	Line 137 to 149
Assembly:	Line 652 to 728

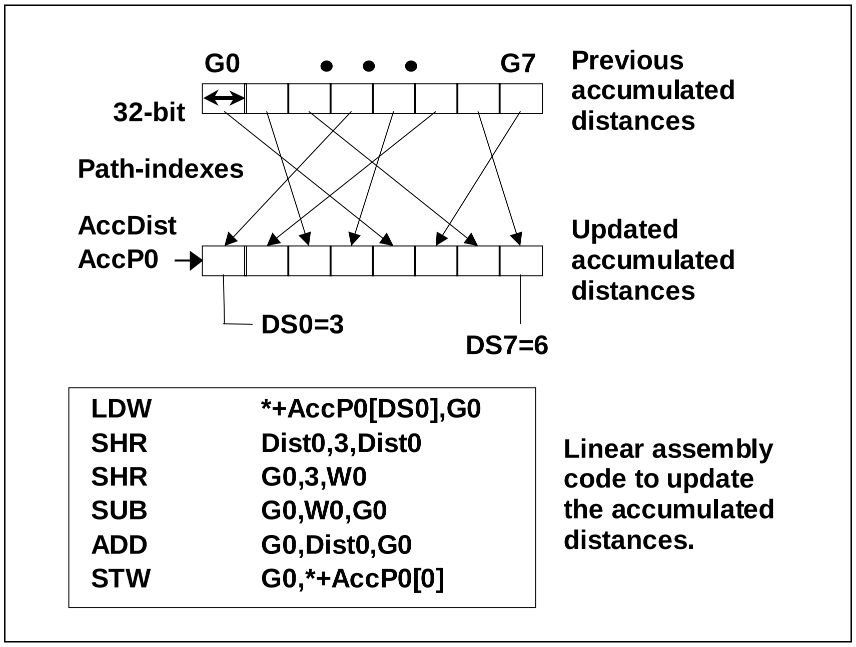
This step accumulates the Dist0 and Dist1 values into the AccDist array, which is an array of eight distances. The following formula summarizes this step.

```
AccDist[i] = AccDist[i] x 7/8 + Dist0 / 8; i = even
AccDist[i] = AccDist[i] x 7/8 + Dist1 / 8; i = 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


G0
• • •
G7
Previous accumulated distances
32-bit
Path-indexes
AccDist
AccP0
Updated accumulated distances
DS0=3
DS7=6
Linear assembly code to update the accumulated distances.

```
LDW     *+AccP0[DS0],G0
SHR     Dist0,3,Dist0
SHR     G0,3,W0
SUB     G0,W0,G0
ADD     G0,Dist0,G0
STW     G0,*+AccP0[0]
```

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.

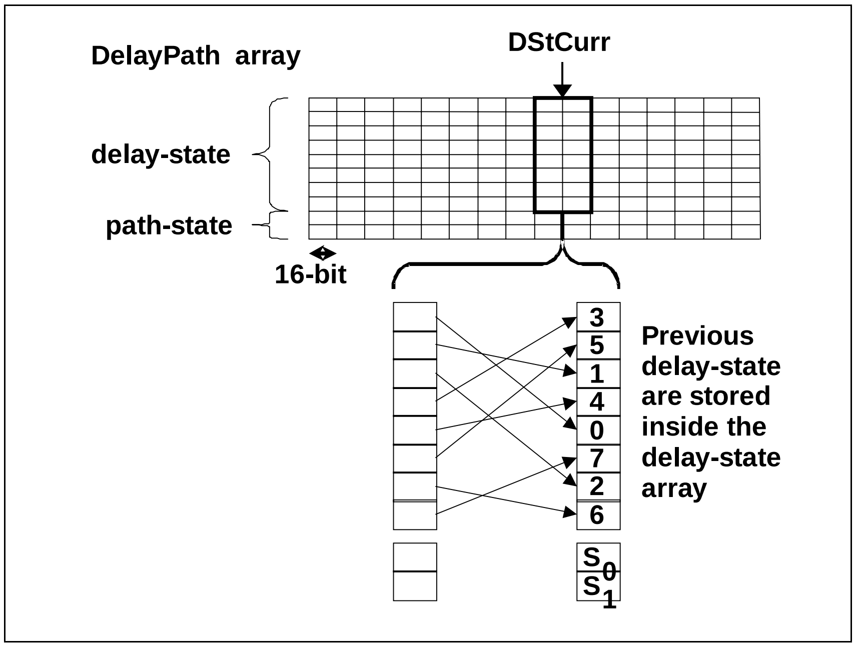
Step 5. Trace Backward for Path-State

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.

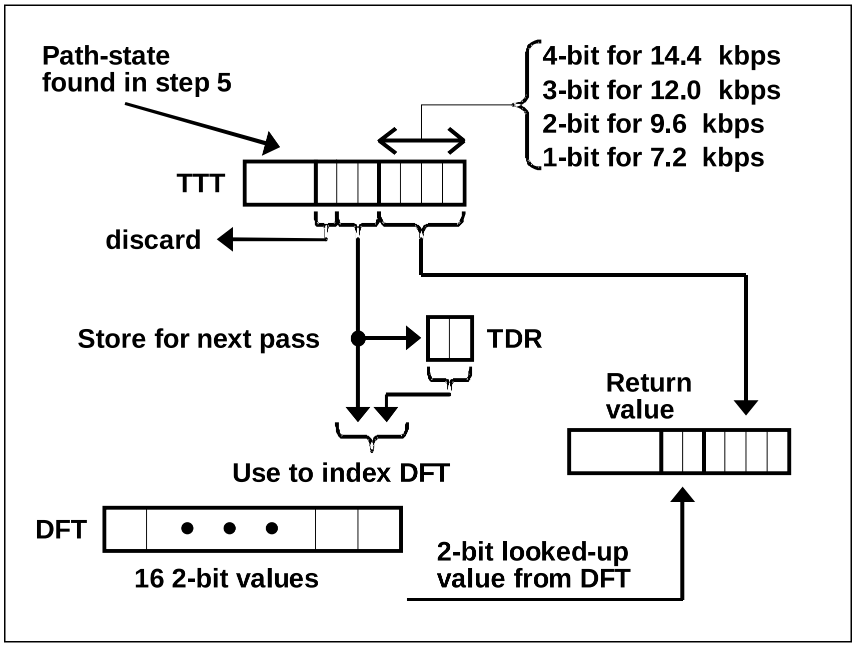
```
              MVK     DelayPath,JJ0
              MVKH    DelayPath,JJ0
              MV      DStCurr,JJ
              MVK     15,II
IILoop:       LDH     *+JJ[IDD],IDD
              CMPGT   JJ,JJ0,TT0
      [TT0]   SUBAH   JJ,10,JJ
      [!TT0]  MVK     DelayPath+300,JJ
      [!TT0]  MVKH    DelayPath+300,JJ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

```
/****************************************************************************
* File:     viterbic.c
* Purpose:  Viterbi in C code.
****************************************************************************/

#include "main.h"

/* Differentiator truth table.  Bit 4,5 for transmitter, bit 0,1 receiver. */
char Diff[] = { 0x00, 0x11, 0x23, 0x32, 0x11, 0x00, 0x32, 0x23,
                0x22, 0x33, 0x10, 0x01, 0x33, 0x22, 0x01, 0x10 };

/* Delay states transition matrix looking backward from present state. */
char TMBackward[8][4] = {
  0, 3, 1, 2,
  4, 7, 6, 5,
  1, 2, 0, 3,
  5, 6, 7, 4,
  2, 1, 3, 0,
  7, 4, 5, 6,
  3, 0, 2, 1,
  6, 5, 4, 7
};

char   Tdr, DelayState[16][8], PathState[16][2], DStCurr;
int    AccDist[8];

/****************************************************************************
* Init - initialization.
****************************************************************************/

void Init ()
{
  short i, j;
  DStCurr = 0;
  Tdr = 0;
  AccDist[0] = 0;
  for (i = 1; i < 8; i++)
    AccDist[i] = (1 << 10) / 2;
  for (j = 0; j < 16; j++)
    for (i = 0; i < 8; i++)
      DelayState[j][i] = 0;
  for (j = 0; j < 16; j++)
    for (i = 0; i < 2; i++)
      PathState[j][i] = 0;
}

/****************************************************************************
* Viterbi - decoder.
****************************************************************************/

int Viterbi (int Xi, int Yi, int modd)
{
  short State[8], Xb, Yb, Nx, Ny, Qx, Qy, Ix, Iy, Id;
  short c, i, j, tmp1, tmp2, tmp3, tmp4, tt, ss, pp;
  int   Distance[8], dist1, dist0, DataOut, Sht;
  Mod   *Mcurr;

  Mcurr = (Mod*) modd;
  Sht = Mcurr->TSht;

  /* With Xi and Yi as input, first get the shortest distances */

  for (c = 0; c < 8; c++)
  {
    Xb = (Mcurr->TXs[c]) << 10;  Yb = (Mcurr->TYs[c]) << 10;
    Nx = Mcurr->TNx[c];          Ny = Mcurr->TNy[c];
    Qx = (Mcurr->TQx) << 10;     Qy = (Mcurr->TQy) << 10;
    Iy = Ny;
    for (i = 0; i < Ny; i++)
    {
      if (Yi < Yb) { Iy = i; break; }
      Yb += Qy;
    }
    Ix = Nx;
    for (i = 0; i < Nx; i++)
    {
      if (Xi < Xb) { Ix = i; break; }
      Xb += Qx;
    }
    Id = Iy * (Nx + 1) + Ix;
    State[c] = *(Mcurr->St[c] + Id);
    tmp1 = *(Mcurr->Xc[c] + Id);
    tmp2 = *(Mcurr->Yc[c] + Id);
    if (State[c] != SS)
    {
      tmp3 = Xi - (tmp1 << 10);
      tmp4 = Yi - (tmp2 << 10);
      Distance[c] = (int)tmp3 * (int)tmp3 + (int)tmp4 * (int)tmp4;
    }
    else
    {
      tmp3 = Xi - (*(Mcurr->Xc[c] + tmp1) << 10);
      tmp4 = Yi - (*(Mcurr->Yc[c] + tmp1) << 10);
      dist1 = (int)tmp3 * (int)tmp3 + (int)tmp4 * (int)tmp4;
      tmp3 = Xi - (*(Mcurr->Xc[c] + tmp2) << 10);
      tmp4 = Yi - (*(Mcurr->Yc[c] + tmp2) << 10);
      Distance[c] = (int)tmp3 * (int)tmp3 + (int)tmp4 * (int)tmp4;
      State[c] = *(Mcurr->St[c] + tmp2);
      if (dist1 < Distance[c])
      {
        Distance[c] = dist1;
        State[c] = *(Mcurr->St[c] + tmp1);
      }
    }
  }

  /* Find the shortest distance amoung path state 0-3 and 4-7 */

  dist0 = 0x7FFFFFFF;
  for (c = 0; c < 4; c++)
  {
    if (Distance[c] < dist0)
    {
      dist0 = Distance[c];
      Id = c;
  } }
  DelayState[DStCurr][0] = TMBackward[0][Id];
  DelayState[DStCurr][2] = TMBackward[2][Id];
  DelayState[DStCurr][4] = TMBackward[4][Id];
  DelayState[DStCurr][6] = TMBackward[6][Id];
  PathState[DStCurr][0] = State[Id];

  dist1 = 0x7FFFFFFF;
  for (c = 4; c < 8; c++)
  {
    if (Distance[c] < dist1)
    {
      dist1 = Distance[c];
      Id = c;
  } }
  DelayState[DStCurr][1] = TMBackward[1][Id-4];
  DelayState[DStCurr][3] = TMBackward[3][Id-4];
  DelayState[DStCurr][5] = TMBackward[5][Id-4];
  DelayState[DStCurr][7] = TMBackward[7][Id-4];
  PathState[DStCurr][1] = State[Id];

  /* Update accumulated distance */

  for (i = 0; i < 8; i++)
    Distance[i] = AccDist[i];
  for (i = 0; i < 8; i++)
  {
    Id = DelayState[DStCurr][i];
    AccDist[i] = Distance[Id] / 8 * 7;
  }
  for (i = 0; i < 8; i+=2)
    AccDist[i] += dist0 / 8;
  for (i = 1; i < 8; i+=2)
    AccDist[i] += dist1 / 8;

  /* Trace backward to get output */

  dist0 = 0x7FFFFFFF;
  for (i = 0; i < 8; i++)
  {
    if (AccDist[i] < dist0)
    {
      dist0 = AccDist[i];
      Id = i;
  } }
  j = DStCurr;
  for (i = 0; i < 15; i++)
  {
    Id = DelayState[j][Id];
    j--; if (j < 0) j = 15;
  }
  tt = PathState[j][Id % 2];
  DStCurr++;
  if (DStCurr >= 16) DStCurr = 0;

  /* Differentiate */

  ss = tt & ((1 << Sht) - 1);
  tt = (tt >> Sht) & 0x03;
  pp = (tt << 2) | Tdr;
  Tdr = tt;
  DataOut = ((Diff[pp] & 0x03) << Sht) | ss;

  return (DataOut);
}

/**********************************  End  ***********************************/
```

Appendix B. V.32Bis Viterbi C62XX Assembly Implementation

```
****************************************************************************
*
*   TEXAS INSTRUMENTS, INC.
*
*   V.32BIS VITERBI DECODING
*
*   FILE:  Viterbi.asm
*
*   REVISION DATE:  12/08/97
*
*   USAGE: This routine is C callable and can be called as
*
*       Data = int Viterbi (int Xi, int Yi, int* Mod);
*
*       Xi = Symbol location in "real" axis, scaled up by 1024.
*       Yi = Symbol location in "imaginary" axis, scaled up by 1024.
*       Mod = Modulation table to use.
*       Data = return symbol data value.
*
*   DESCRIPTION:  A complementary C code and a complete calling example code
*                 should be included with this file.
*
****************************************************************************

****************************************************************************
* Variables declaration.
****************************************************************************

****** Global and static variables:
        .bss    Tdr,4,4             ; Differentiator Tdr value.
        .bss    DStCurrbss,4,4      ; Current delay state pointer.
        .bss    DelayPath,320,4     ; DelayState and PathState array in half.
        .bss    AccDist,32,4        ; Accumulated distance array in word.

****** Variables that are temporary and local:
        .bss    Distance,32,4       ; Distance Result array in word.
        .bss    Dummy,2,2           ; To allow simaltaneous access.
        .bss    State,16,4          ; State array in half.
        .bss    MCurr,4,4           ; Save current modulation pointer.

****** Constant declaration:
SK      .set    1024                ; Scale factor.

****************************************************************************
* Lookup Tables.
****************************************************************************

        .sect   "LUTable"

****** Modulation tables:

MMMMM   .set    0x7FFF          ; Set to maximum (for modulation table use)
SS      .set    0xFFFF          ; Unused state value (for mod table use)
SZ      .set    3               ; Data point word size (for mod table use)

        .def    _M14            ; Make this table visible to outside.
_M14:                           ; Modulation table for 14.4 kbps.
        .int    M140            ; First address.
        .int    0x35E           ; Extract bit 4 - 5 (The Y bits in V.32bis)
        .int    0x39C           ; Extract bit 0 - 3 (The Q bits in V.32bis)
        .int    4               ; Shift amount (Number of Q bits in V.32bis)

```

```
M140    .int    M141                                      ; Next address
        .short  -5*SK,-6*SK,4*SK,4*SK,(3+1)*SZ,(4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -7*SK,-4*SK,-3*SK,-8*SK,0x05,0x00       ; Y0,X0,Y1,X1,S0,S1
        .short  -7*SK,-4*SK,MMMMM,MMMMM,0x05,SS
        .short  -7*SK,+0*SK,MMMMM,MMMMM,0x07,SS
        .short  -7*SK,+4*SK,MMMMM,MMMMM,0x03,SS
        .short  -7*SK,+4*SK,-3*SK,+8*SK,0x03,0x01
        .short  -3*SK,-8*SK,MMMMM,MMMMM,0x00,SS
        .short  -3*SK,-4*SK,MMMMM,MMMMM,0x04,SS
        .short  -3*SK,+0*SK,MMMMM,MMMMM,0x06,SS
        .short  -3*SK,+4*SK,MMMMM,MMMMM,0x02,SS
        .short  -3*SK,+8*SK,MMMMM,MMMMM,0x01,SS
        .short  +1*SK,-8*SK,MMMMM,MMMMM,0x08,SS
        .short  +1*SK,-4*SK,MMMMM,MMMMM,0x0C,SS
        .short  +1*SK,+0*SK,MMMMM,MMMMM,0x0E,SS
        .short  +1*SK,+4*SK,MMMMM,MMMMM,0x0A,SS
        .short  +1*SK,+8*SK,MMMMM,MMMMM,0x09,SS
        .short  +5*SK,-4*SK,+1*SK,-8*SK,0x0D,0x08
        .short  +5*SK,-4*SK,MMMMM,MMMMM,0x0D,SS
        .short  +5*SK,+0*SK,MMMMM,MMMMM,0x0F,SS
        .short  +5*SK,+4*SK,MMMMM,MMMMM,0x0B,SS
        .short  +5*SK,+4*SK,+1*SK,+8*SK,0x0B,0x09

M141    .int    M142                                      ; Next address
        .short  -3*SK,-6*SK,4*SK,4*SK,(3+1)*SZ,(4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -5*SK,-4*SK,-1*SK,-8*SK,0x1B,0x19       ; Y0,X0,Y1,X1,S0,S1
        .short  -5*SK,-4*SK,MMMMM,MMMMM,0x1B,SS
        .short  -5*SK,+0*SK,MMMMM,MMMMM,0x1F,SS
        .short  -5*SK,+4*SK,MMMMM,MMMMM,0x1D,SS
        .short  -5*SK,+4*SK,-1*SK,+8*SK,0x1D,0x18
        .short  -1*SK,-8*SK,MMMMM,MMMMM,0x19,SS
        .short  -1*SK,-4*SK,MMMMM,MMMMM,0x1A,SS
        .short  -1*SK,+0*SK,MMMMM,MMMMM,0x1E,SS
        .short  -1*SK,+4*SK,MMMMM,MMMMM,0x1C,SS
        .short  -1*SK,+8*SK,MMMMM,MMMMM,0x18,SS
        .short  +3*SK,-8*SK,MMMMM,MMMMM,0x11,SS
        .short  +3*SK,-4*SK,MMMMM,MMMMM,0x12,SS
        .short  +3*SK,+0*SK,MMMMM,MMMMM,0x16,SS
        .short  +3*SK,+4*SK,MMMMM,MMMMM,0x14,SS
        .short  +3*SK,+8*SK,MMMMM,MMMMM,0x10,SS
        .short  +7*SK,-4*SK,+3*SK,-8*SK,0x13,0x11
        .short  +7*SK,-4*SK,MMMMM,MMMMM,0x13,SS
        .short  +7*SK,+0*SK,MMMMM,MMMMM,0x17,SS
        .short  +7*SK,+4*SK,MMMMM,MMMMM,0x15,SS
        .short  +7*SK,+4*SK,+3*SK,+8*SK,0x15,0x10

M142    .int    M143                                      ; Next address
        .short  -7*SK,-4*SK,4*SK,4*SK,(4+1)*SZ,(3+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -9*SK,-2*SK,-5*SK,-6*SK,0x28,0x2D       ; Y0,X0,Y1,X1,S0,S1
        .short  -9*SK,-2*SK,MMMMM,MMMMM,0x28,SS
        .short  -9*SK,+2*SK,MMMMM,MMMMM,0x20,SS
        .short  -9*SK,+2*SK,-5*SK,+6*SK,0x20,0x25
        .short  -5*SK,-6*SK,MMMMM,MMMMM,0x2D,SS
        .short  -5*SK,-2*SK,MMMMM,MMMMM,0x2C,SS
        .short  -5*SK,+2*SK,MMMMM,MMMMM,0x24,SS
        .short  -5*SK,+6*SK,MMMMM,MMMMM,0x25,SS
        .short  -1*SK,-6*SK,MMMMM,MMMMM,0x2F,SS
        .short  -1*SK,-2*SK,MMMMM,MMMMM,0x2E,SS
        .short  -1*SK,+2*SK,MMMMM,MMMMM,0x26,SS
        .short  -1*SK,+6*SK,MMMMM,MMMMM,0x27,SS
        .short  +3*SK,-6*SK,MMMMM,MMMMM,0x2B,SS
        .short  +3*SK,-2*SK,MMMMM,MMMMM,0x2A,SS
        .short  +3*SK,+2*SK,MMMMM,MMMMM,0x22,SS
        .short  +3*SK,+6*SK,MMMMM,MMMMM,0x23,SS
        .short  +7*SK,-2*SK,+3*SK,-6*SK,0x29,0x2B
        .short  +7*SK,-2*SK,MMMMM,MMMMM,0x29,SS
        .short  +7*SK,+2*SK,MMMMM,MMMMM,0x21,SS
        .short  +7*SK,+2*SK,+3*SK,+6*SK,0x21,0x23

M143    .int    M144                                ; Next address
        .short  -5*SK,-4*SK,4*SK,4*SK,(4+1)*SZ,(3+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -7*SK,-2*SK,-3*SK,-6*SK,0x31,0x33   ; Y0,X0,Y1,X1,S0,S1
        .short  -7*SK,-2*SK,MMMMM,MMMMM,0x31,SS
        .short  -7*SK,+2*SK,MMMMM,MMMMM,0x39,SS
        .short  -7*SK,+2*SK,-3*SK,+6*SK,0x39,0x3B
        .short  -3*SK,-6*SK,MMMMM,MMMMM,0x33,SS
        .short  -3*SK,-2*SK,MMMMM,MMMMM,0x32,SS
        .short  -3*SK,+2*SK,MMMMM,MMMMM,0x3A,SS
        .short  -3*SK,+6*SK,MMMMM,MMMMM,0x3B,SS
        .short  +1*SK,-6*SK,MMMMM,MMMMM,0x37,SS
        .short  +1*SK,-2*SK,MMMMM,MMMMM,0x36,SS
        .short  +1*SK,+2*SK,MMMMM,MMMMM,0x3E,SS
        .short  +1*SK,+6*SK,MMMMM,MMMMM,0x3F,SS
        .short  +5*SK,-6*SK,MMMMM,MMMMM,0x35,SS
        .short  +5*SK,-2*SK,MMMMM,MMMMM,0x34,SS
        .short  +5*SK,+2*SK,MMMMM,MMMMM,0x3C,SS
        .short  +5*SK,+6*SK,MMMMM,MMMMM,0x3D,SS
        .short  +9*SK,-2*SK,+5*SK,-6*SK,0x30,0x35
        .short  +9*SK,-2*SK,MMMMM,MMMMM,0x30,SS
        .short  +9*SK,+2*SK,MMMMM,MMMMM,0x38,SS
        .short  +9*SK,+2*SK,+5*SK,+6*SK,0x38,0x3D

M144    .int    M145                                ; Next address
        .short  -4*SK,-5*SK,4*SK,4*SK,(3+1)*SZ,(4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -6*SK,-3*SK,-2*SK,-7*SK,0x4B,0x49   ; Y0,X0,Y1,X1,S0,S1
        .short  -6*SK,-3*SK,MMMMM,MMMMM,0x4B,SS
        .short  -6*SK,+1*SK,MMMMM,MMMMM,0x4F,SS
        .short  -6*SK,+5*SK,MMMMM,MMMMM,0x4D,SS
        .short  -6*SK,+5*SK,-2*SK,+9*SK,0x4D,0x48
        .short  -2*SK,-7*SK,MMMMM,MMMMM,0x49,SS
        .short  -2*SK,-3*SK,MMMMM,MMMMM,0x4A,SS
        .short  -2*SK,+1*SK,MMMMM,MMMMM,0x4E,SS
        .short  -2*SK,+5*SK,MMMMM,MMMMM,0x4C,SS
        .short  -2*SK,+9*SK,MMMMM,MMMMM,0x48,SS
        .short  +2*SK,-7*SK,MMMMM,MMMMM,0x41,SS
        .short  +2*SK,-3*SK,MMMMM,MMMMM,0x42,SS
        .short  +2*SK,+1*SK,MMMMM,MMMMM,0x46,SS
        .short  +2*SK,+5*SK,MMMMM,MMMMM,0x44,SS
        .short  +2*SK,+9*SK,MMMMM,MMMMM,0x40,SS
        .short  +6*SK,-3*SK,+2*SK,-7*SK,0x43,0x41
        .short  +6*SK,-3*SK,MMMMM,MMMMM,0x43,SS
        .short  +6*SK,+1*SK,MMMMM,MMMMM,0x47,SS
        .short  +6*SK,+5*SK,MMMMM,MMMMM,0x45,SS
        .short  +6*SK,+5*SK,+2*SK,+9*SK,0x45,0x40

M145    .int    M146                                ; Next address
        .short  -4*SK,-7*SK,4*SK,4*SK,(3+1)*SZ,(4+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -6*SK,-5*SK,-2*SK,-9*SK,0x55,0x50   ; Y0,X0,Y1,X1,S0,S1
        .short  -6*SK,-5*SK,MMMMM,MMMMM,0x55,SS
        .short  -6*SK,-1*SK,MMMMM,MMMMM,0x57,SS
        .short  -6*SK,+3*SK,MMMMM,MMMMM,0x53,SS
        .short  -6*SK,+3*SK,-2*SK,+7*SK,0x53,0x51
        .short  -2*SK,-9*SK,MMMMM,MMMMM,0x50,SS
        .short  -2*SK,-5*SK,MMMMM,MMMMM,0x54,SS
        .short  -2*SK,-1*SK,MMMMM,MMMMM,0x56,SS
        .short  -2*SK,+3*SK,MMMMM,MMMMM,0x52,SS
        .short  -2*SK,+7*SK,MMMMM,MMMMM,0x51,SS
        .short  +2*SK,-9*SK,MMMMM,MMMMM,0x58,SS
        .short  +2*SK,-5*SK,MMMMM,MMMMM,0x5C,SS
        .short  +2*SK,-1*SK,MMMMM,MMMMM,0x5E,SS
        .short  +2*SK,+3*SK,MMMMM,MMMMM,0x5A,SS
        .short  +2*SK,+7*SK,MMMMM,MMMMM,0x59,SS
        .short  +6*SK,-5*SK,+2*SK,-9*SK,0x5D,0x58
        .short  +6*SK,-5*SK,MMMMM,MMMMM,0x5D,SS
        .short  +6*SK,-1*SK,MMMMM,MMMMM,0x5F,SS
        .short  +6*SK,+3*SK,MMMMM,MMMMM,0x5B,SS
        .short  +6*SK,+3*SK,+2*SK,+7*SK,0x5B,0x59

M146    .int    M147                                    ; Next address
        .short  -6*SK,-5*SK,4*SK,4*SK,(4+1)*SZ,(3+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -8*SK,-3*SK,-4*SK,-7*SK,0x61,0x63       ; Y0,X0,Y1,X1,S0,S1
        .short  -8*SK,-3*SK,MMMMM,MMMMM,0x61,SS
        .short  -8*SK,+1*SK,MMMMM,MMMMM,0x69,SS
        .short  -8*SK,+1*SK,-4*SK,+5*SK,0x69,0x6B
        .short  -4*SK,-7*SK,MMMMM,MMMMM,0x63,SS
        .short  -4*SK,-3*SK,MMMMM,MMMMM,0x62,SS
        .short  -4*SK,+1*SK,MMMMM,MMMMM,0x6A,SS
        .short  -4*SK,+5*SK,MMMMM,MMMMM,0x6B,SS
        .short  +0*SK,-7*SK,MMMMM,MMMMM,0x67,SS
        .short  +0*SK,-3*SK,MMMMM,MMMMM,0x66,SS
        .short  +0*SK,+1*SK,MMMMM,MMMMM,0x6E,SS
        .short  +0*SK,+5*SK,MMMMM,MMMMM,0x6F,SS
        .short  +4*SK,-7*SK,MMMMM,MMMMM,0x65,SS
        .short  +4*SK,-3*SK,MMMMM,MMMMM,0x64,SS
        .short  +4*SK,+1*SK,MMMMM,MMMMM,0x6C,SS
        .short  +4*SK,+5*SK,MMMMM,MMMMM,0x6D,SS
        .short  +8*SK,-3*SK,+4*SK,-7*SK,0x60,0x65
        .short  +8*SK,-3*SK,MMMMM,MMMMM,0x60,SS
        .short  +8*SK,+1*SK,MMMMM,MMMMM,0x68,SS
        .short  +8*SK,+1*SK,+4*SK,+5*SK,0x68,0x6D

M147    .int    M140                                    ; Next address
        .short  -6*SK,-3*SK,4*SK,4*SK,(4+1)*SZ,(3+1)*SZ ; Ys,Xs,Yq,Xq,Ny+1,Nx+1
        .short  -8*SK,-1*SK,-4*SK,-5*SK,0x78,0x7D       ; Y0,X0,Y1,X1,S0,S1
        .short  -8*SK,-1*SK,MMMMM,MMMMM,0x78,SS
        .short  -8*SK,+3*SK,MMMMM,MMMMM,0x70,SS
        .short  -8*SK,+3*SK,-4*SK,+7*SK,0x70,0x75
        .short  -4*SK,-5*SK,MMMMM,MMMMM,0x7D,SS
        .short  -4*SK,-1*SK,MMMMM,MMMMM,0x7C,SS
        .short  -4*SK,+3*SK,MMMMM,MMMMM,0x74,SS
        .short  -4*SK,+7*SK,MMMMM,MMMMM,0x75,SS
        .short  +0*SK,-5*SK,MMMMM,MMMMM,0x7F,SS
        .short  +0*SK,-1*SK,MMMMM,MMMMM,0x7E,SS
        .short  +0*SK,+3*SK,MMMMM,MMMMM,0x76,SS
        .short  +0*SK,+7*SK,MMMMM,MMMMM,0x77,SS
        .short  +4*SK,-5*SK,MMMMM,MMMMM,0x7B,SS
        .short  +4*SK,-1*SK,MMMMM,MMMMM,0x7A,SS
        .short  +4*SK,+3*SK,MMMMM,MMMMM,0x72,SS
        .short  +4*SK,+7*SK,MMMMM,MMMMM,0x73,SS
        .short  +8*SK,-1*SK,+4*SK,-5*SK,0x79,0x7B
        .short  +8*SK,-1*SK,MMMMM,MMMMM,0x79,SS
        .short  +8*SK,+3*SK,MMMMM,MMMMM,0x71,SS
        .short  +8*SK,+3*SK,+4*SK,+7*SK,0x71,0x73

        .def    _M12            ; Make this table visible to outside.
_M12:                           ; Modulation table for 12.0 kbps.
        ; . . .
        ; . . .

****** Transition matrix backward table:

TMBack  .int    0x03122130      ; 2D array index:  43 42 41 40 03 02 01 00
        .int    0x65475674      ; (data in 4-bit)  53 52 51 50 13 12 11 10
        .int    0x12033021      ;                  63 62 61 60 23 22 21 20
        .int    0x74564765      ;                  73 72 71 70 33 32 31 30

****** Receiver differentiator table:

Diff    .int    0x1B4EE1B4      ; index (data in 2-bit):  F E D C ... 2 1 0

*****************************************************************************
```

```
* Initialization of the variables.
*****************************************************************************

        .text
        .def    _Init
_Init:

****** DStCurrbss = *DelayPath.
        MVK     DStCurrbss,A0
        MVKH    DStCurrbss,A0
        MVK     DelayPath,A1
        MVKH    DelayPath,A1
        STW     A1,*A0

****** DelayPath[i][j] = 0, i from 0 to 15, j from 0 to 9.
        MVK     16,B0
PLoop:
        MVK     10,B1
        ZERO    A0
KLoop:
        SUB     10,B1,A3
        STH     A0,*+A1[A3]
  [B1]  SUB     B1,1,B1
  [B1]  B       KLoop
        NOP     5
        ADDAH   A1,10,A1
  [B0]  SUB     B0,1,B0
  [B0]  B       PLoop
        NOP     5

****** AccDist[i] = 512, i from 1 to 7.
****** AccDist[0] = 0.
        MVK     SK/2,A0
        MVK     AccDist,A1
        MVKH    AccDist,A1
        MVK     8,B0
ILoop:
        SUB     8,B0,A3
        STW     A0,*+A1[A3]
  [B0]  SUB     B0,1,B0
  [B0]  B       ILoop
        NOP     5
        ZERO    A0
        STW     A0,*+A1[0]

****** Tdr = 0.
        ZERO    A0
        MVK     Tdr,A1
        MVKH    Tdr,A1
        STW     A0,*A1

****** Return.
        B       B3
        NOP     5

*****************************************************************************
* Viterbi Program section.
*****************************************************************************

*** Registers name that will be used for all the Steps.
DStCurr .set    A13     ; Current delay state pointer.
TT0     .set    A1      ; Temporary for testing purpose.
TT1     .set    B1      ; Temporary for testing purpose.
Q0      .set    A3      ; Temporary with a handy name.
Q1      .set    B3      ; Temporary with a handy name.
W0      .set    A6      ; Temporary with a handy name.
W1      .set    B6      ; Temporary with a handy name.
```

```

*** Registers name that will be used for in Step 3 and Step 4.
DS0     .set    A0      ; Delay state 0
DS1     .set    B0      ; Delay state 1
DS2     .set    A1      ; Delay state 2
DS3     .set    B1      ; Delay state 3
DS4     .set    A2      ; Delay state 4
DS5     .set    B2      ; Delay state 5
DS6     .set    A3      ; Delay state 6
DS7     .set    B3      ; Delay state 7
Dist0   .set    A7      ; Distance 0
Dist1   .set    B7      ; Distance 1

*** Registers name that will be used for in Step 4 and Step 5.
G0      .set    A5      ; Accumulate distance 0.
G1      .set    B5      ; Accumulate distance 1.
G2      .set    A10     ; Accumulate distance 2.
G3      .set    B10     ; Accumulate distance 3.
G4      .set    A11     ; Accumulate distance 4.
G5      .set    B11     ; Accumulate distance 5.
G6      .set    A9      ; Accumulate distance 6.
G7      .set    B9      ; Accumulate distance 7.

*** Registers name that will be used for in Step 5 and Step 6.
TTT     .set    A8      ; Traced back path state result.

*** Program start label.
            .def    _Viterbi
            .text
_Viterbi:

*** Step 1 - Opening statements.        ;*
***          Purpose - Allocate stack to store B3, A10-A13, and B10-B13.
***                    Place argunment 3 (A6) to MCurr (bss).
***                    Setup current delay state pointer to DStCurr.

            MV      A6,B6               ;*
  ||        LDW     *A6,MCurr0          ;** Get first modulation table.
  ||        SUBAW   B15,9,B15           ;* Allocate storage.

            MVK     MCurr,B0            ;*
  ||        MVK     DStCurrbss,A1       ;*
  ||        MV      B15,A9              ;* Set up a A register save faster.
  ||        STW     B3,*+B15[1]         ;* Save registers.

            MVKH    MCurr,B0            ;*
  ||        MVKH    DStCurrbss,A1       ;*
  ||        STW     A13,*+A9[8]         ;* Save registers.
  ||        STW     B13,*+B15[9]        ;* Save registers.

            STW     B6,*B0              ;* Save current pointer first.
  ||        LDW     *A1,DStCurr         ;* Get Delay state pointer.

            STW     A12,*+A9[6]         ;* Save registers.
  ||        STW     B12,*+B15[7]        ;* Save registers.

*** Step 2 - Distances and states.      ;**
***          Purpose - Calculate the Distance and State value
***                    based on the input Xi and Yi.
***          Input: Xi, Yi (registers)
***          Output: Distance, State (bss)

XY      .set    A0      ; X and Y for SUB2 operation.
Cc      .set    B0      ; Loop counter.
Xt      .set    A1      ; Test X
Yt      .set    B1      ; Test Y
S1      .set    A1      ; State 1
```

```
S0      .set    B1      ; State 0
TT      .set    B2      ; Temporary
Nx      .set    A3      ; Nx
Ny      .set    B3      ; Ny
Xi      .set    A4      ; Symbol received - Argunment 1
Yi      .set    B4      ; Symbol received - Argunment 2
Xs      .set    A5      ; Xs
Ys      .set    B5      ; Ys
MCurr0  .set    A6      ; Bit rate modulation table to use
StateP  .set    B6      ; Points to state result
Xq      .set    A7      ; Xq
Yq      .set    B7      ; Yq
DistP   .set    A8      ; Points to distance result
MCurr1  .set    B8      ; Table pointer for B register file
Ix      .set    A9      ; X index
Iy      .set    B9      ; Y index
Id0     .set    A9      ; Region result
Id1     .set    B9      ; Region result for B register file
P2      .set    A9      ; Product register
P3      .set    B9      ; Product register
T0      .set    A10     ; Temporary register
T1      .set    B10     ; Temporary register
P0      .set    A10     ; Product register
P1      .set    B10     ; Product register
T2      .set    A11     ; Temporary register
T3      .set    B11     ; Temporary register
MNext0  .set    A12     ; Next table pointer

Step2:
            MV      MCurr0,TT           ;**
  ||        MVK     State,StateP        ;** Set up State result pointer.
  ||        MVK     Distance,DistP      ;** Set up Distance result pointer.
  ||        STW     A11,*+A9[4]         ;* Save registers.
  ||        STW     B11,*+B15[5]        ;* Save registers.

            LDH     *+MCurr0[3],Xs      ;** Get Xs and Ys from table.
  ||        LDH     *+TT[2],Ys          ;**
  ||        MVKH    State,StateP        ;**
  ||        MVKH    Distance,DistP      ;**

            LDH     *+MCurr0[5],Xq      ;** Get Xq and Yq from table.
  ||        LDH     *+TT[4],Yq          ;**

            LDH     *+MCurr0[7],Nx      ;** Get Nx and Ny from table.
  ||        LDH     *+TT[6],Ny          ;**
  ||        ZERO    Ix                  ;** Initialize indexes.
  ||        ZERO    Iy                  ;**
  ||        MVK     8,Cc                ;** Initialize loop counter.

            STW     A10,*+A9[2]         ;* Save registers.
  ||        STW     B10,*+B15[3]        ;* Save registers.

            NOP     2                   ;**

Cloop:
            CMPGT   Xi,Xs,Xt            ;** Compare and increment test level.
  ||        ADD     Xs,Xq,Xs            ;**
  ||        CMPGT   Yi,Ys,Yt            ;**
  ||        ADD     Ys,Yq,Ys            ;**
  ||        LDW     *MCurr0,MNext0      ;**^ Prepare for next iteration.
  ||        MV      MCurr0,TT           ;**

            .loop   4                   ;** Set to Max (Nx, Ny).
            CMPGT   Xi,Xs,Xt            ;** Compare and increment test level.
  || [Xt]   ADD     Xs,Xq,Xs            ;**
  || [Xt]   ADD     Ix,3,Ix             ;** Also increase indexes.
  ||        CMPGT   Yi,Ys,Yt            ;**
  || [Yt]   ADD     Ys,Yq,Ys            ;**
  || [Yt]   ADD     Iy,Nx,Iy            ;**
            .endloop                    ;**

            ADD     Ix,Iy,Id0           ;** Calculate region.
  ||        ADD     Ix,Iy,Id1           ;**
  ||        ADDAH   MCurr0,8,MCurr0     ;** Points to path state start.
  ||        ADDAH   TT,10,MCurr1        ;** Pointer for B register file.

            LDW     *MCurr0[Id0],T0     ;** Get X0, Y0.
  ||        LDW     *MCurr1[Id1],T1     ;** Get X1, Y1.
  ||        ADD     Id0,2,Id0           ;**

            LDW     *MCurr0[Id0],T2     ;** Get State 0 and 1.

            SHL     Xi,16,XY            ;** Combine Xi Yi to XY.
  ||        CLR     Yi,16,31,TT         ;**
  ||        LDH     *+MNext0[2],Ys      ;**^ Prepare for next iteration.

            OR      XY,TT,XY            ;** Combine Xi Yi to XY.
  ||        LDH     *+MNext0[3],Xs      ;**^ Prepare for next iteration.

            LDH     *+MNext0[4],Yq      ;**^ Prepare for next iteration.

            SUB2    T0,XY,T0            ;** Get X Y difference from X0 Y0.
  ||        SUB2    T1,XY,T1            ;** Get X Y difference from X1 Y1.
  ||        LDH     *+MNext0[5],Xq      ;**^ Prepare for next iteration.

            MPY     T0,T0,P2            ;** Get (Yi - Y0) ^ 2.
  ||        MPY     T1,T1,P3            ;** Get (Yi - Y1) ^ 2.
  ||        LDH     *+MNext0[6],Ny      ;**^ Prepare for next iteration.
  || [Cc]   SUB     Cc,1,Cc             ;** Decrement loop counter.

            MPYH    T0,T0,P0            ;** Get (Xi - X0) ^ 2.
  ||        MPYH    T1,T1,P1            ;** Get (Yi - Y0) ^ 2.
  ||        LDH     *+MNext0[7],Nx      ;**^ Prepare for next iteration.
  || [Cc]   B       Cloop               ;** Next iteration.

            NOP                         ;** Delay slot for multiply.

            ADD     P0,P2,T0            ;** Get distance square from X0 Y0.
  ||        ADD     P1,P3,T1            ;** Get distance square from X1 Y1.
  ||        MV      T2,T3               ;** State 0 and 1 info here.

            CMPGT   T0,T1,TT            ;** Which distance is larger.
  ||        EXTU    T3,16,16,S0         ;** Separate state 0 and 1.
  ||        EXTU    T2,0,16,S1          ;**

     [TT]   MV      T1,T0               ;** If distance from X0 Y0 larger.
  || [TT]   MV      S1,S0               ;**    then use distance X1 Y1.

            STW     T0,*DistP++         ;** Save results.
  ||        STH     S0,*StateP++        ;**
  ||        MV      MNext0,MCurr0       ;**^ Prepare for next iteration.
  ||        ZERO    Ix                  ;**^ Reset indexes.
  ||        ZERO    Iy                  ;**^
            ; Branch to Cloop here.

*** Step 3 - Shortest distance          ;***
***          Purpose - Find the shortest distances and place it in Dist0
***                    and Dist1.  From the distance indexes, get the delay
***                    states and store them.
***          Input: Distance, State (bss)
***          Output: Dist0, Dist1 (registers)
***                  DS0 - DS7 (registers)
***                  DelayPath (bss)
```

```
TMB0    .set    A0      ; Table pointer
TMB1    .set    B0      ; Table pointer
T200A   .set    A2      ; Store 200 hex
T200B   .set    B2      ; Store 200 hex
DistP0  .set    A4      ; Distance pointer
DistP1  .set    B4      ; Distance pointer
Ix0     .set    A5      ; Index
Ix1     .set    B5      ; Index
DSP0    .set    A6      ; Delay state pointer
DSP1    .set    B6      ; Delay state pointer
Stp0    .set    A8      ; State pointer 0
Stp1    .set    B8      ; State pointer 1
St0     .set    A9      ; State 0
St1     .set    B9      ; State 1
LU0     .set    A10     ; Lookup table row 0
LU1     .set    B10     ; Lookup table row 1
LU2     .set    A11     ; Lookup table row 2
LU3     .set    B11     ; Lookup table row 3

Step3:
            MVK     Distance,DistP0     ;*** Points to Distance 0 to 3.
  ||        MVK     Distance,DistP1     ;*** Points to Distance 4 to 7.

            MVKH    Distance,DistP0     ;***
  ||        MVKH    Distance,DistP1     ;***

            LDW     *+DistP0[0],Dist0   ;*** Get Distance 0.
  ||        LDW     *+DistP1[4],Dist1   ;*** Get Distance 4.
  ||        MVK     TMBack,TMB0         ;*** Get table pointer.
  ||        MVK     TMBack,TMB1         ;*** Get table pointer.

            LDW     *+DistP0[1],Q0      ;*** Get Distance 1.
  ||        LDW     *+DistP1[5],Q1      ;*** Get Distance 5.
  ||        MVKH    TMBack,TMB0         ;***
  ||        MVKH    TMBack,TMB1         ;***

            LDW     *+TMB0[0],LU0       ;*** Read lookup table.
  ||        LDW     *+TMB1[1],LU1       ;*** Read lookup table.
  ||        MVK     State,Stp0          ;*** Get state pointer.
  ||        MVK     State,Stp1          ;*** Get state pointer.

            LDW     *+DistP0[2],Q0      ;*** Get Distance 2.
  ||        LDW     *+DistP1[6],Q1      ;*** Get Distance 6.
  ||        MVKH    State,Stp0          ;*** Get state pointer.
  ||        MVKH    State,Stp1          ;*** Get state pointer.

            LDW     *+TMB0[2],LU2       ;*** Read lookup table.
  ||        LDW     *+TMB1[3],LU3       ;*** Read lookup table.

            MVK     0x200,T200A         ;***
  ||        MVK     0x200,T200B         ;***
  ||        LDW     *+DistP0[3],Q0      ;*** Get Distance 3.
  ||        LDW     *+DistP1[7],Q1      ;*** Get Distance 7.

            CMPGT   Dist0,Q0,TT0        ;*** If Distance 1 < lowest.
  ||        MVK     0x39C,Ix0           ;*** Extract bit 0 - 3.
  ||        CMPGT   Dist1,Q1,TT1        ;*** If Distance 5 < lowest.
  ||        MVK     0x39C,Ix1           ;*** Extract bit 0 - 3.
  ||        LDH     *Stp0[0],St0        ;***
  ||        LDH     *Stp1[4],St1        ;***

      [TT0] MV      Q0,Dist0            ;*** Keep lowest distances.
  ||  [TT0] MVK     0x31C,Ix0           ;*** Extract bit 4 - 7.
  ||  [TT0] LDH     *Stp0[1],St0        ;***
  ||  [TT1] MV      Q1,Dist1            ;*** Keep lowest distances.
  ||  [TT1] MVK     0x31C,Ix1           ;*** Extract bit 4 - 7.
  ||  [TT1] LDH     *Stp1[5],St1        ;***
```

```

          CMPGT   Dist0,Q0,TT0        ;*** If Distance 2 < lowest.
   ||     CMPGT   Dist1,Q1,TT1        ;*** If Distance 6 < lowest.

      [TT0]  MV   Q0,Dist0            ;*** Keep lowest distances.
   || [TT0]  MVK  0x29C,Ix0           ;*** Extract bit 8 - 11.
   || [TT0]  LDH  *Stp0[2],St0        ;***
   || [TT1]  MV   Q1,Dist1            ;*** Keep lowest distances.
   || [TT1]  MVK  0x29C,Ix1           ;*** Extract bit 8 - 11.
   || [TT1]  LDH  *Stp1[6],St1        ;***

          CMPGT   Dist0,Q0,TT0        ;*** If Distance 3 < lowest.
   ||     CMPGT   Dist1,Q1,TT1        ;*** If Distance 7 < lowest.
   ||     MV      DStCurr,DSP0        ;*** Initialize pointer for A side.
   ||     MV      DStCurr,DSP1        ;*** Initialize pointer for B side.

      [TT0]  MV   Q0,Dist0            ;*** Keep lowest distances.
   || [TT0]  MVK  0x21C,Ix0           ;*** Extract bit 12 - 15.
   || [TT0]  LDH  *Stp0[3],St0        ;***
   || [TT1]  MV   Q1,Dist1            ;*** Keep lowest distances.
   || [TT1]  MVK  0x21C,Ix1           ;*** Extract bit 12 - 15.
   || [TT1]  LDH  *Stp1[7],St1        ;***

          EXTU    LU0,Ix0,DS0         ;*** Do the extraction.
   ||     EXTU    LU1,Ix1,DS1         ;*** Do the extraction.

          EXTU    LU2,Ix0,DS2         ;*** Do the extraction.
   ||     EXTU    LU3,Ix1,DS3         ;*** Do the extraction.
   ||     SUB     Ix0,T200A,Ix0       ;*** Extract range is upper 16 bit.
   ||     SUB     Ix1,T200B,Ix1       ;*** Extract range is upper 16 bit.
   ||     STH     DS0,*+DSP0[0]       ;*** Save to DelayState 0.
   ||     STH     DS1,*+DSP1[1]       ;*** Save to DelayState 1.

          EXTU    LU0,Ix0,DS4         ;*** Do the extraction.
   ||     EXTU    LU1,Ix1,DS5         ;*** Do the extraction.
   ||     STH     DS2,*+DSP0[2]       ;*** Save to DelayState 2.
   ||     STH     DS3,*+DSP1[3]       ;*** Save to DelayState 3.

          EXTU    LU2,Ix0,DS6         ;*** Do the extraction.
   ||     EXTU    LU3,Ix1,DS7         ;*** Do the extraction.
   ||     STH     DS4,*+DSP0[4]       ;*** Save to DelayState 4.
   ||     STH     DS5,*+DSP1[5]       ;*** Save to DelayState 5.

          STH     DS6,*+DSP0[6]       ;*** Save to DelayState 6.
   ||     STH     DS7,*+DSP1[7]       ;*** Save to DelayState 7.
   ||     MVK     AccDist,AccP0       ;****
   ||     MVK     AccDist,AccP1       ;****

          STH     St0,*+DSP0[8]       ;*** Save Path-state info.
   ||     STH     St1,*+DSP1[9]       ;*** Save Path-state info.
   ||     MVKH    AccDist,AccP0       ;****
   ||     MVKH    AccDist,AccP1       ;****

*** Step 4 - Accumulate distance      ;****
***          Purpose - Calculate the accumulated distances using a fix formula.
***          Input: DS0 - DS7 (registers)
***                 Dist0, Dist1 (registers)
***          Output: AccDist (bss)
***                  G0 - G7 (registers)

AccP0  .set    A4      ; Accumulate Distance pointer
AccP1  .set    B4      ; Accumulate Distance pointer

Step4:
          LDW     *+AccP0[DS0],G0     ;**** Read accumulate distance.
   ||     LDW     *+AccP1[DS1],G1     ;****

```

```
            LDW     *+AccP0[DS2],G2     ;**** Read accumulate distance.
  ||        LDW     *+AccP1[DS3],G3     ;****

            LDW     *+AccP0[DS4],G4     ;**** Read accumulate distance.
  ||        LDW     *+AccP1[DS5],G5     ;****

            LDW     *+AccP0[DS6],G6     ;**** Read accumulate distance.
  ||        LDW     *+AccP1[DS7],G7     ;****
  ||        MVK     1,ONE               ;***** ONE = 1.

            SHR     Dist0,3,Dist0       ;**** Divide Distances by 8.
  ||        SHR     Dist1,3,Dist1       ;****

            SHR     G0,3,W0             ;**** Divide accumulate distances by 8.
  ||        SHR     G1,3,W1             ;****

            SUB     G0,W0,G0            ;**** Subtract to get 7/8 of itself.
  ||        SUB     G1,W1,G1            ;****
  ||        SHR     G2,3,W0             ;****^ Divide accumulate distance by 8.
  ||        SHR     G3,3,W1             ;****^
  ||        MPY     ONE,0,Q0            ;***** Q0 index at 0.
  ||        MPY     ONE,1,Q1            ;***** Q1 index at 1.

            ADD     G0,Dist0,G0         ;**** Add the distances.
  ||        ADD     G1,Dist1,G1         ;****
  ||        SUB     G2,W0,G2            ;****^ Subtract to get 7/8 of itself.
  ||        SUB     G3,W1,G3            ;****^
  ||        SHR     G4,3,W0             ;****^^ Divide accumulate distance by 8.
  ||        SHR     G5,3,W1             ;****^^

            STW     G0,*+AccP0[0]       ;**** Store accumulate distances.
  ||        STW     G1,*+AccP1[1]       ;****
  ||        ADD     G2,Dist0,G2         ;****^ Add the distances.
  ||        ADD     G3,Dist1,G3         ;****^
  ||        SUB     G4,W0,G4            ;****^^ Subtract to get 7/8 of itself.
  ||        SUB     G5,W1,G5            ;****^^

            STW     G2,*+AccP0[2]       ;****^ Store accumulate distances.
  ||        STW     G3,*+AccP1[3]       ;****^
  ||        ADD     G4,Dist0,G4         ;****^^ Add the distances.
  ||        ADD     G5,Dist1,G5         ;****^^
  ||        SHR     G6,3,W0             ;****^^^ Divide accumulate distance by 8.
  ||        SHR     G7,3,W1             ;****^^^

            STW     G4,*+AccP0[4]       ;****^^ Store accumulate distances.
  ||        STW     G5,*+AccP1[5]       ;****^^
  ||        SUB     G6,W0,G6            ;****^^^ Subtract to get 7/8 of itself.
  ||        SUB     G7,W1,G7            ;****^^^
  ||        CMPGT   G0,G2,TT0           ;***** Find out which is smaller.
  ||        CMPGT   G1,G3,TT1           ;*****

            ADD     G6,Dist0,G6         ;****^^^ Add the distances.
  ||        ADD     G7,Dist1,G7         ;****^^^
  || [TT0]  MV      G2,G0               ;***** Save smaller.
  || [TT0]  MPY     ONE,2,Q0            ;***** Update index.
  || [TT1]  MV      G3,G1               ;***** Save smaller.
  || [TT1]  MPY     ONE,3,Q1            ;***** Update index.

            STW     G6,*+AccP0[6]       ;****^^^ Store accumulate distances.
  ||        STW     G7,*+AccP1[7]       ;****^^^
  ||        CMPGT   G0,G1,TT0           ;***** Find out which is smaller.
  ||        MPY     ONE,4,W0            ;***** W0 index at 4.
  ||        MPY     ONE,5,W1            ;***** W1 index at 5.

*** Step 5 - Backward trace             ;*****
***         Purpose - First find the smallest accumulated distance, then
***                   trace backwrd using DelayPath to find the path-state.
***                    Increment the delay state pointer DStCurrbss.
***          Input: G0 - G7 (registers)
***                 DelayPath (bss)
***          Output: TTT (registers)

II      .set    B0      ; Temporary.
IDD     .set    A4      ; Index.
DStptr  .set    A5      ; Delay State pointer.
JJ      .set    A7      ; Temporary.
JJ0     .set    B8      ; Temporary.
ONE     .set    A12     ; One

Step5:
     [TT0]  MV      G1,G0               ;***** Minimum of first four found.
  || [!TT0] MV      Q0,Q1               ;***** Value in G0, index in Q1.
  ||        CMPGT   G4,G6,TT0           ;***** Find minimum of next four.
  ||        CMPGT   G5,G7,TT1           ;*****

     [TT0]  MV      G6,G4               ;***** Save smaller.
  || [TT0]  MPY     ONE,6,W0            ;***** Update index.
  || [TT1]  MV      G7,G5               ;***** Save smaller.
  || [TT1]  MPY     ONE,7,W1            ;***** Update index.
  ||        MVK     MCurr,MCurrx        ;****** Get back pointer.

            CMPGT   G4,G5,TT0           ;***** Find out which is smaller.
  ||        MVKH    MCurr,MCurrx        ;******

     [TT0]  MV      G5,G4               ;***** Minimum of last four found.
  || [!TT0] MV      W0,W1               ;***** Value in G4, index in W1.
  ||        MVK     DelayPath,JJ0       ;***** Prepare for circular buffer for JJ.

            CMPGT   G0,G4,TT0           ;***** Get to find the real minimum.
  ||        MV      Q1,IDD              ;***** Place index.
  ||        MVKH    DelayPath,JJ0       ;***** Prepare for circular buffer for JJ.

     [TT0]  MV      W1,IDD              ;***** Got index at IDD.
  ||        MV      DStCurr,JJ          ;***** Get JJ for trace back.
  ||        MVK     14,II               ;***** Trace back counter.
  ||        LDW     *MCurrx,MCurrx      ;****** Load the pointer.

IILoop:
      [II]  B       IILoop              ;***** Trace back.

            LDH     *+JJ[IDD],IDD       ;***** Keep tracing.
  ||        MVK     Diff,DFTP           ;****** Get diff table pointer.

            CMPGT   JJ,JJ0,TT0          ;***** Do JJ circular buffer.
  ||        MVKH    Diff,DFTP           ;****** Get diff table pointer.

     [TT0]  SUBAH   JJ,10,JJ            ;***** Adjust JJ in circular fashion.
  || [!TT0] MVK     DelayPath+300,JJ    ;***** Adjust JJ in circular fashion.

     [!TT0] MVKH    DelayPath+300,JJ    ;***** Adjust JJ in circular fashion.

            SUB     II,1,II             ;***** Decrement counter.
            ; Branch to IILoop here.

            AND     IDD,1,IDD           ;***** LSB decides which path-state to pick.
  ||        LDW     *+MCurrx[1],Ext1    ;****** Get extraction indexes.
  ||        MVK     Tdr,TDRP            ;****** Get Tdr pointer.

            ADD     8,IDD,IDD           ;***** Index it to path-state in DelayPath.
  ||        LDW     *+MCurrx[2],Ext2    ;****** Get extraction indexes.
  ||        MVKH    Tdr,TDRP            ;****** Get Tdr pointer.

            LDH     *+JJ[IDD],TTT       ;***** Get the path-state.
  ||        MVK     DelayPath+300,JJ0   ;***** Prepare for cir buffer for DStCurr.
  ||        MVK     DStCurrbss,DStptr   ;***** Get DStCurr pointer.

            MVKH    DelayPath+300,JJ0   ;***** Prepare for cir buffer for DStCurr.
  ||        MVKH    DStCurrbss,DStptr   ;***** Get DStCurr pointer.
  ||        LDW     *TDRP,TDR           ;****** Read Tdr.

            CMPLT   DStCurr,JJ0,TT0     ;***** Do DStCurr circular buffer.
  ||        LDW     *+MCurrx[3],Sht     ;****** Get shift indexes.

     [TT0]  ADDAH   DStCurr,10,DStCurr  ;***** Adjust DStCurr in circular fashion.
  || [!TT0] MVK     DelayPath,DStCurr   ;***** Adjust DStCurr in circular fashion.
  ||        LDW     *+B15[1],B3         ;******* Return sequences.

     [!TT0] MVKH    DelayPath,DStCurr   ;***** Adjust DStCurr in circular fashion.
  ||        LDW     *DFTP,DFT           ;****** Read diff table.

*** Step 6 - Differentiate              ;******
***          Purpose - Differentiate the result.
***          Input: TTT (registers)
***          Output: A4 (registers) return value.

MCurrx  .set    A0      ; Modulation table pointer.
PP      .set    A0      ; Temporary.
Ext2    .set    A2      ; Extraction index 2.
TDR     .set    B2      ; Tdr value.
SSS     .set    A3      ; Temporary.
Sht     .set    A4      ; Shift amount.
TDRP    .set    B4      ; Tdr pointer.
Ext1    .set    A6      ; Extraction index 1.
DFT     .set    A7      ; Diff table value.
DFTP    .set    A9      ; Diff table pointer.

Step6:
            STW     DStCurr,*DStptr     ;***** Save DStCurr.
  ||        EXTU    TTT,Ext1,PP         ;****** Separate TTT into two pieces.

            SHL     PP,2,PP             ;****** Get diff table index.
  ||        STW     PP,*TDRP            ;****** Update new Tdr value.

            OR      PP,TDR,PP           ;****** Get diff table index.
  ||        EXTU    TTT,Ext2,SSS        ;****** Separate TTT into two pieces.

            SHL     PP,1,PP             ;****** Get diff table index.
  ||        MV      B15,A9              ;******* Return sequences.
  ||        B       B3                  ;******* Return sequences.

            SHR     DFT,PP,DFT          ;****** Differentiate.
  ||        LDW     *+A9[2],A10         ;******* Return sequences.
  ||        LDW     *+B15[3],B10        ;******* Return sequences.

            CLR     DFT,2,31,DFT        ;****** Keep lower 2 bits.
  ||        LDW     *+A9[4],A11         ;******* Return sequences.
  ||        LDW     *+B15[5],B11        ;******* Return sequences.

            SHL     DFT,Sht,DFT         ;****** Combine with SSS.
  ||        LDW     *+A9[6],A12         ;******* Return sequences.
  ||        LDW     *+B15[7],B12        ;******* Return sequences.

            OR      DFT,SSS,A4          ;****** Combine with SSS.
  ||        LDW     *+A9[8],A13         ;******* Return sequences.
  ||        LDW     *+B15[9],B13        ;******* Return sequences.

            ADDAW   B15,9,B15           ;******* Deallocate storage.
            ; Branch back to calling routine here.

*** Step 7 - Return                     ;*******
***          Purpose - Restore registers A10 - A13, B10 - B13, and A3.
***                     Jump back to calling routine.
Step7:
            .end

```

Appendix C. Main Program to Test the Algorithm

```
/*****************************************************************************
* File:     main.c
* Purpose:  main C program to test the performance of the Viterbi code.
*****************************************************************************/

#include "main.h"

/*****************************************************************************
* Trellis - encoder.
*****************************************************************************/

/* Delay states transition matrix looking forward from present state. */
char TMForward[8][4] = {
  0, 6, 2, 4,
  2, 4, 0, 6,
  4, 2, 6, 0,
  6, 0, 4, 2,
  1, 5, 7, 3,
  3, 7, 5, 1,
  7, 3, 1, 5,
  5, 1, 3, 7
};

extern Mod M14C;
Mod    *Mcurr = &M14C;
char   Sht = 4;
char   Tdt = 0, TransmitState = 0;

void Trellis (int DataIn, int *Xo, int *Yo)
{
  /* Differentiator truth table.  Bit 4,5 for transmitter, bit 0,1 receiver. */
  char Diff[] = { 0x00, 0x11, 0x23, 0x32, 0x11, 0x00, 0x32, 0x23,
                  0x22, 0x33, 0x10, 0x01, 0x33, 0x22, 0x01, 0x10 };
  short  tt, ss, pp;
  tt = (DataIn >> Sht) & 0x03;
  tt = (tt << 2) | Tdt;
  tt = (Diff[tt] >> 4) & 0x03;
  Tdt = tt;
  pp = tt | (TransmitState & 0x04);
  TransmitState = TMForward[TransmitState][tt];
  ss = DataIn & ((1 << Sht) - 1);
  ss = *(Mcurr->Pt[pp] + ss);
  *Xo = *(Mcurr->Xc[pp] + ss) << 10;
  *Yo = *(Mcurr->Yc[pp] + ss) << 10;
}

/*****************************************************************************
* ReadData - setup test data.
*****************************************************************************/

int ReadData ()
{
  int DataIn;
  static short i = 0;
  static short TestData[] = {
    0x01, 0x11, 0x31, 0x01, 0x11, 0x31, 0x01, 0x11, 0x31, 0x00
  };
  DataIn = TestData[i];
  i++; if (i >= 10) i = 0;
  return (DataIn);
}

/*****************************************************************************
* OutData - output test data.
```

```
****************************************************************************/

void OutData (int DataOut)
{
  /* ?? = DataOut; */
}

/****************************************************************************
* main - Run test.
****************************************************************************/

extern int M14;
#define Mod14 &M14

int Viterbi (int, int, int*);
int Init (void);

main ()
{
  int count = 0;
  int DataIn, DataOut, Xo, Yo, Xi, Yi;
  Init ();
  for (;;)
  {
    DataIn = ReadData();
    Trellis (DataIn, &Xo, &Yo);
    Xi = Xo;  Yi = Yo;
#ifdef AA
    DataOut = Viterbi (Xi,Yi,Mod14);        /* For assembly Viterbi */
#endif
#ifdef CC
    DataOut = Viterbi (Xi,Yi,(int*)&M14C);  /* For C Viterbi        */
#endif
    OutData (DataOut);
    count++;
  }
}
```