U.S. Digital Cellular Error-Correction Coding Algorithm Implementation on the TMS320C5x

Application Report

Mansoor A. Chishtie
Digital Signal Processing Applications — Semiconductor Group

SPRA137
October 1994

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 applications 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.

Abstract

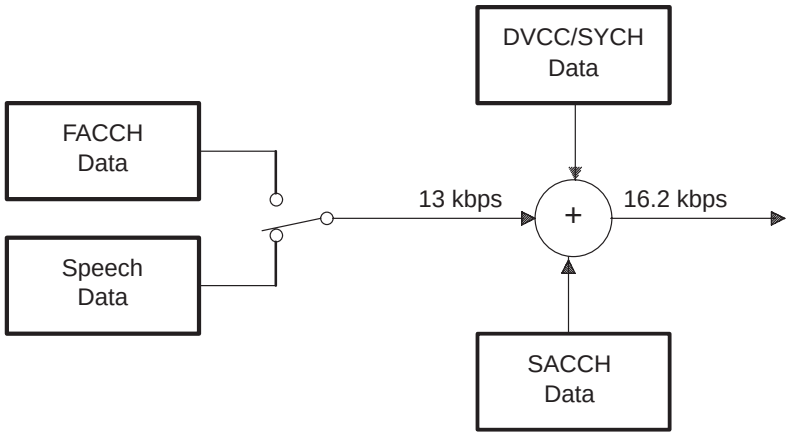
Programmable digital signal processors are commonly used in U.S. digital cellular terminal designs. All digital cellular transmitters employ convolutional and CRC codes to protect against channel-induced errors. Receivers typically use Viterbi decoders and CRC syndrome checks to verify that the decoded data contains no errors. This paper presents selected implementation examples of the error-protection and correction functions of various cellular data channels using the TMS320C5x digital signal processor family.

Introduction

Programmable DSPs are widely used in the new U.S. digital cellular (USDC) radio designs. The primary function of the DSPs in these designs is baseband signal processing. However, many designs are also using the newer DSPs as the system coordinator in the radio, a task typically performed by a microcontroller. This trend is caused by a) system care-abouts of low cost, low power, and small form factor, and b) newer generations of DSPs (such as the TI TMS320C5x family) that have architectures suitable for microcontroller-type functions.

One of the several signal-processing-intensive tasks that a digital cellular radio needs to perform is error protection and correction. The IS-54 voice channels transmit voice and control information in digital form. Although these radio links are primarily used for digital voice transmission (VSELP), a portion of the channel capacity is reserved for control information. This relatively slow bit-rate link is used for background control information such as broadcast messages, mobile-assisted handoffs, etc. This is called slow associated control channel (SACCH) in IS-54 terminology. Another type of signaling channel is called fast associated control channel (FACCH). However, FACCH messages are not sent simultaneously with the voice data. They replace the compressed voice data whenever necessary. Figure 1 shows how these messages are multiplexed with voice data.

Figure 1. Voice and Control-Channel Multiplexing Over One Time Slot



These three digital data channels employ extensive error-protection and correction mechanisms to protect all or most of the transmitted information. Convolutional codes, CRC codes, and bit/frame interleaving techniques are used for this purpose. The convolutional coding schemes used by these three channels are not identical and require slightly different decoding methods to be employed by the receivers. Despite these

minor differences, the basic decoding algorithm used by the three channels is usually a Viterbi algorithm. In the rest of this paper, these channel formats are explained separately, a suitable decoding scheme is presented, and its implementation details are discussed.

VSELP Channel Format

The VSELP encoder compresses the digitized speech from 64 kbps to 7.950 kbps. Additional information is added for error protection to increase the total data transfer rate to 13 kbps. The VSELP algorithm operates on a frame-by-frame basis in which each speech frame is 20 ms in duration. The VSELP encoder generates 159 bits of compressed speech for each speech frame. These bits are grouped into two classes: 77 class-I bits that need error protection and 82 class-II bits that are sent without any error protection. Class-I bits are protected from channel-induced errors by applying convolutional encoding. Furthermore, error detection is also provided by applying a 7-bit CRC code to the 12 most perceptually significant class-I bits. Finally, this 260-bit speech frame is interleaved over two time slots to protect against burst errors.

Figure 2. Error Protection for VSELP Data

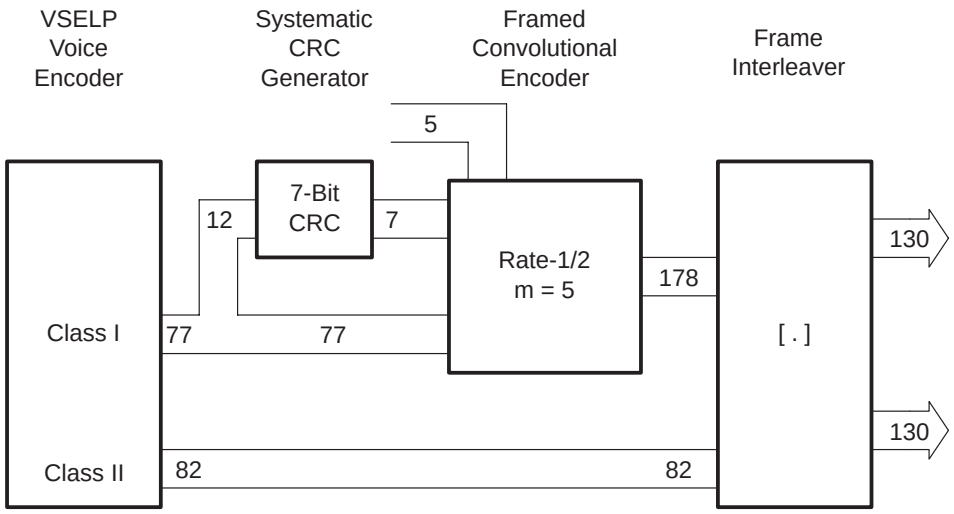
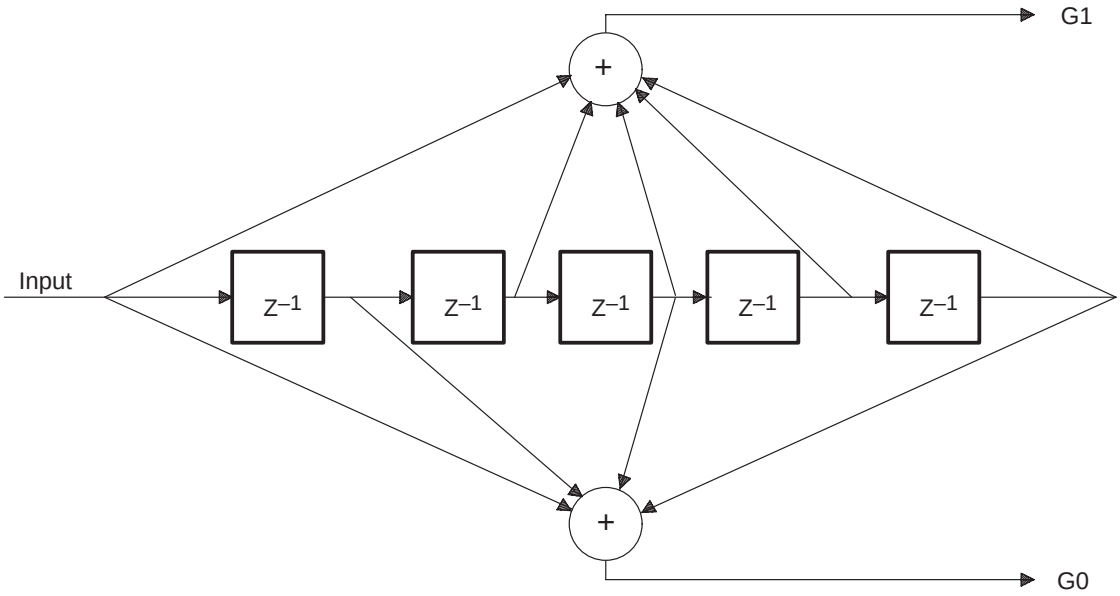
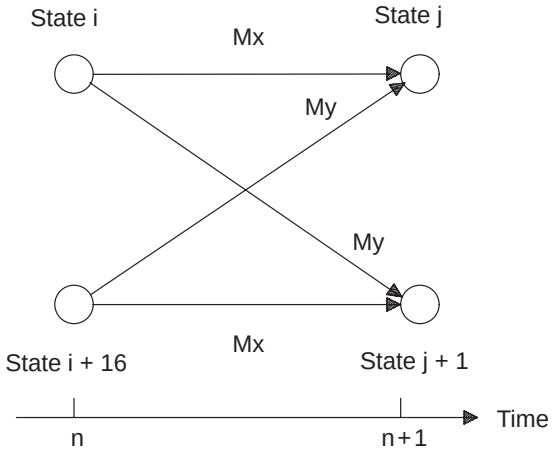


Figure 3. Convolutional Encoder for VSELP Data



The VSELP convolutional encoder is a rate-1/2 framed encoder with a constraint length (denoted by M) of 5. The frame size is 89 bits (see Figure 2), which consists of 77 class-I bits, a 7-bit CRC, and 5 tail bits. Both the initial and the final states are 0. The trellis diagram for this encoder consists of 32 states (that is, 2^M) with each state in symbol interval n connected to two states in the next time interval n + 1. The basic building block of this trellis is shown in Figure 4.

Figure 4. Representative Trellis Section for VSELP Convolutional Encoder



Due to the rate-1/2 encoding scheme, each state is linked to two states in the previous time interval, as shown in Figure 4. The Viterbi algorithm operates on received data by expanding the trellis over a frame length of 89 symbol intervals. Refer to [4] and [5] for general Viterbi algorithm descriptions. A 32-element accumulated cost metric is set up where each element corresponds to one state. Each link from an old state

to a new state has a transition cost associated with it. For instance, Mx is the transition cost from state i to state j in Figure 4. These transition costs, which are computed at the symbol rate, reflect the current channel conditions. Each transition cost indicates the probability of a state i to state j transition over one symbol interval. Consider Figure 4 where state j at time interval n + 1 can be reached from either state i or state i + 16 in time interval n. The Viterbi algorithm selects the more likely transition into state j by comparing the total accumulated cost of the two possible links. The accumulated cost of each link is computed by adding the current transition cost to its previous accumulated cost. For example, new accumulated costs of the two links entering state j in Figure 4 are:

$$\text{new\_acc\_cost}[j] = \text{old\_acc\_cost}[i] + Mx \quad \textbf{(1)}$$

$$\text{new\_acc\_cost}[j] = \text{old\_acc\_cost}[i + 16] + My \quad \textbf{(2)}$$

The smaller of the two values is selected and the corresponding link is retained for further processing in the next time interval. The other candidate is discarded. This process of selecting one transition entering a state is performed on all 32 states at each symbol interval. Path history of every state is maintained for the entire 89-symbol-long frame. When one frame is processed completely, the state 0 in the last time interval is selected, and its associated path is considered the most likely received path. This path is traced to find the most likely received bit sequence. As shown in Figure 2, the encoder pads five tail bits (all 0s) at the end of each message frame. This ensures that the last encoder state is always 0. Additionally, the initial state of the encoder is also 0 by definition. This requires special consideration by the decoder during initialization of the accumulated cost metric at the beginning of each frame. To assure that state 0 is selected by the algorithm at the beginning of each frame, it is initialized with a lower cost value than that of the other 31 states.

This algorithm can be implemented more efficiently if the underlying symmetry of the trellis structure used is considered. As shown in Figure 4, a pair of states in a symbol interval are connected to another pair of states in the next interval with no other connections to the rest of the trellis. Therefore, all state transitions during one symbol interval can be uniquely broken down into 16 butterfly-like structures similar to Figure 4. Furthermore, only two transition cost values are associated with the four links of each butterfly (Mx and My in Figure 4; in some implementations, Mx is always equal to –My, which leads to further simplification of the structure). This symmetrical structure allows a subroutine that will operate on one butterfly at a time, computing new accumulated cost metrics, selecting the best transition, and storing the path history. This subroutine (or a macro) is invoked 16 times at each symbol interval to update 32 state transitions. Example 1 lists the pseudocode for this function.

Example 1. Pseudocode for Trellis Expansion

```
Acc_Metric1[n]    + Curr_M[x] –> AccB
Acc_Metric1[n+16] + Curr_M[y] –> Acc
min(Acc,AccB) –> Acc_Metric2[m]
If (Acc > AccB) then
   shift 1 in Trans_Tbl[i]
else
   shift 0 in Trans_Tbl[i]
Acc_Metric1[n]    + Curr_M[y] –> AccB
Acc_Metric1[n+16] + Curr_M[x] –> Acc
min(Acc,AccB) –> Acc_Metric2[m+1]
If (Acc > AccB) then
   shift 1 in Trans_Tbl[i+1]
else
   shift 0 in Trans_Tbl[i+1]
```

The pseudocode shown above performs necessary computations for two states, similar to the butterfly structure shown in Figure 4. There are two accumulated cost metrics used by the code, Acc_Metric1[] and Acc_Metric2[]. One contains previous cost metrics and the other is used to store new accumulated cost metrics. At each symbol interval, roles of the two arrays are reversed. Only two array elements need to be accessed by the subroutine. The offsets between those two elements are always 16 and 1 for the two arrays, respectively. This allows for simple indexing of these arrays regardless of which state is currently being accessed. Similarly, only two current metric values Curr_M[] are accessed by the function. The offset between these two elements can also be made equal to 1 if this array is set up in the form of a circular buffer. Finally, since the path history is stored for the two states j and j + 1 in time n + 1, the two elements of the transition table Trans_Tbl[] that need to be accessed are also offset by 1.

Considerable coding efficiency is gained by taking into account these structural symmetries of the trellis butterfly. As shown in pseudocode above, the accumulator and the accumulator buffer are used to hold total accumulated cost of the two links. The TMS320C5x DSPs support special instructions to select the smaller (or larger) of the two values. The CRLT instruction and the conditional-execute instruction (XC) are used in this implementation to select the lower cost link and update the accumulated cost array and the transition table. Since the accumulated cost arrays are accessed only in steps of 1 or 16, indirect addressing modes of postincrement and postmodification by an index of 16 are used to step efficiently through the table. The current transition cost array, Curr_M[], consists of four elements representing four symbols of the rate-1/2 encoder. It is set up as two circular buffers, each containing two elements. In Example 2, the code listing shows the function implemented in ’C5x assembly code. It is set up as a macro that is invoked 16 times to update all 32 states per time interval.

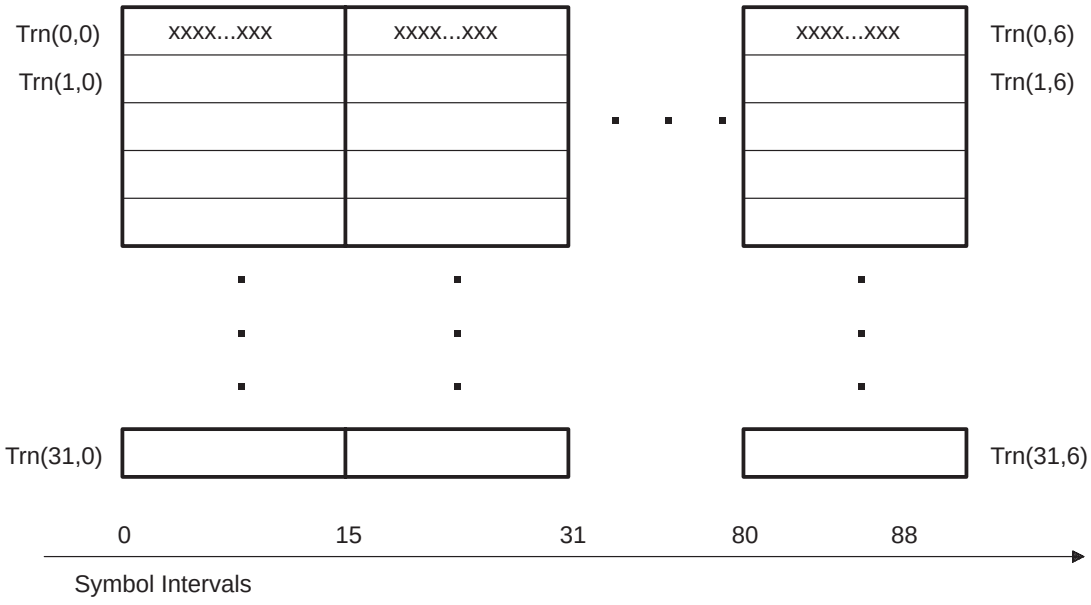
Example 2. Trellis Expansion Macro In ’C5x Assembly Code

```
*
* Entry Conditions:
*
* ARP =  AR1
* INDX=  16
* AR1 –> AccMa[n] ;n=0..31
* CurrMPtr –> CurrM[i] ;i=0..3
* Circ.buffers: CurrM[0..1] and CurrM[2..3]
* AR3 –> AccMb[m] ;m=0..31
* AR4 –> Trn[k]   ;k=0..(6*32)
*
* Exit Conditions:
*
* AR1 –> AccMa[n+1]
* CurrMPtr –> CurrM[i]
* AR3 –> AccMb[m+2]
* AR4 –> Trn[k+2]
*
Texpand .macro  CurrMPtr
        lacc    *0+,CurrMPtr    ; load AccM1[n]
        add     *+,ar1          ; add CurrM[x]
        sacb                    ;
        lacc    *0–,CurrMPtr    ; load AccM1[n+16]
        add     *,ar3           ; add CurrM[y]
        crlt              ; change to crgt for correlation type metric
        sacl    *+,ar4          ; min(path1,path2) –> AccM2[m]
        lacc    *,1             ; load Trn[i]
        xc      1,c             ; if path1>path2
        add     #1              ; shift 1 in Trn[i]
        sacl    *+,ar1          ; save Trn[i]
*
        lacc    *0+,CurrMPtr    ; load AccM1[n]
        add     *+,ar1          ; add CurrM[y]
        sacb
        lacc    *0–,CurrMPtr    ; load AccM1[n+16]
        add     *,ar3           ; add CurrM[x]
        crlt              ; change to crgt for correlation type metric
        sacl    *+,ar4          ; min(path1,path2) –> AccM2[m+1]
        lacc    *,1             ; load Trn[i+1]
        xc      1,c             ; if path1>path2
        add     #1              ; shift 1 in Trn[i+1]
        sacl    *+,ar1          ; save Trn[i+1]
*
        mar     *+
        .endm
```

Path History Memory Organization

Path history is generated by the decoder during the forward pass as it expands the trellis. Given that each encoder state can only be reached from one of the two possible states in the previous symbol interval, a single bit can be used to store this information. The state transition table Trn[x,y] is a 32 × 6 word matrix in which each bit position in a row of elements corresponds to one symbol interval. Each row element in a column corresponds to one of the 32 encoder states. In other words, if Trn[x,y] is the matrix where x = 0 ... 31, and y = 0 ... 5, then x corresponds to the encoder state and (16y + bit position) corresponds to the symbol interval.

Figure 5. Transition Table Organization



Trace-Back

Trace-back starts from state 0 in the 89th symbol interval. The corresponding bit in the transition table indicates which state is linked to it in the 88th symbol interval. This bit is the decoder output in the 89th symbol interval. Next, the decoder jumps to the selected state in the 88th symbol interval, generating the next output bit. This procedure is repeated until all 89 symbol intervals are traced back, producing one frame of decoded output. Example 3 shows this algorithm in pseudo-C code.

Example 3. Trace-Back Function — Pseudo-C Code

```
state = 0;
n = 0;
for (word=6; word>=0; word--) {
   for (bitno=15; bitno>=0; bitno--) {
      if (Trn[state,word].bitno == 0) {
         store 0 in Output[n++];
         state = state>>1;
      }
      else {
         store 1 in Output[n++];
         state = (state>>1) + 16;
      }
   }
}
```

This trace-back function is implemented on the ’C5x using its zero-overhead loop structure. Indirect index addressing is used to step through the transition table efficiently. The INDX register holds the current state ID (0 to 15). Bit-reversed addressing is used to left-shift the INDX register for each iteration. Dynamic bit testing is done by using the TREG2 register as a bit pointer to each element of the transition table. Example 4 lists this ’C5x assembly routine.

Example 4. Trace-Back Implementation in ’C5x Assembly Code

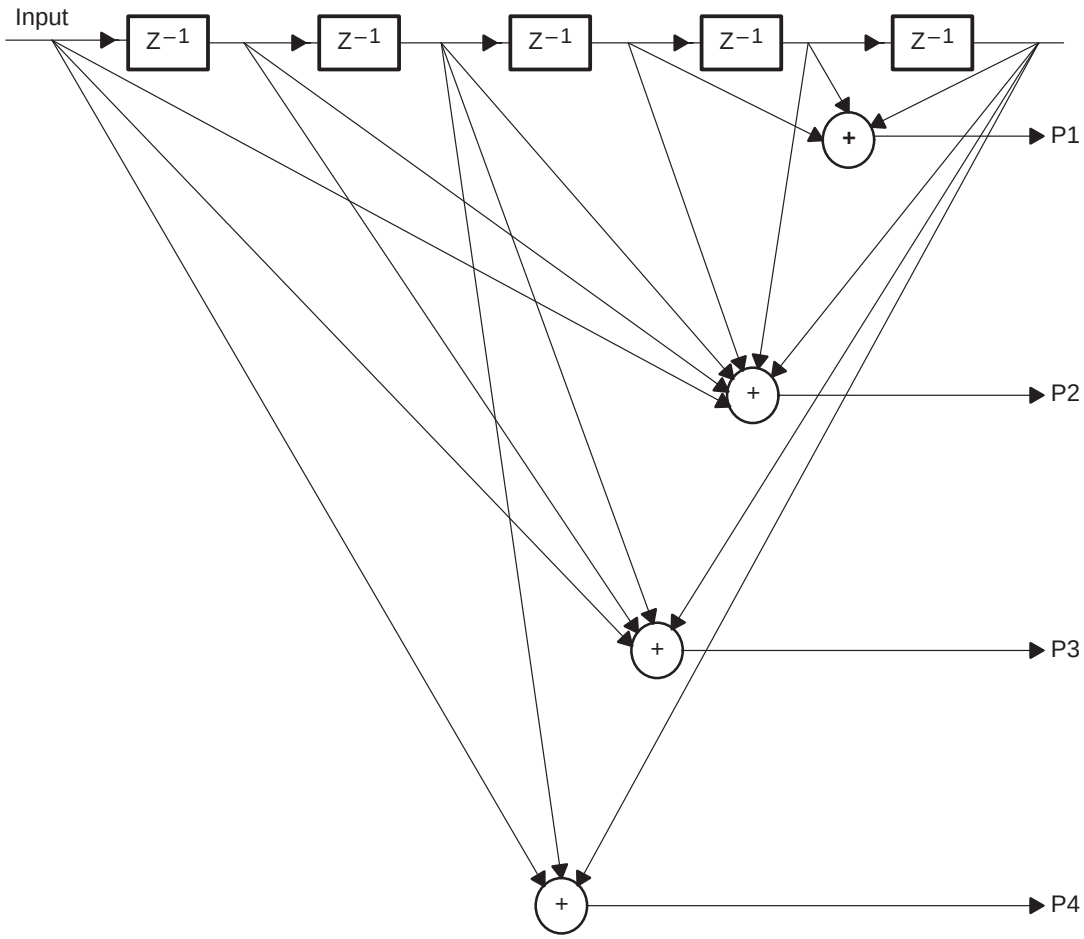
```
TraceBack:
        lar     ar0,#0          ; ar0 is n of Trn[n] (0..31)
        lar     ar3,#OutBuf     ; ar3 –> OutBuf[0]
        lar     ar4,#Trn        ; ar4 –> Path history table
        lacl    #1
        samm    dbmr            ; initialize mask
        lacl    #16–1
        samm    brcr            ; initialize loop count
        lacl    #15
        samm    treg2           ; initialize bit pointer
*
        rptb    trace–1         ; loop 16 times
        mar     *,ar3
        sar     ar0,*
        apl     *+,ar4          ; save OutBuf[i]
        mar     *0+
        bitt    *0–,ar0         ; test bit(i) of Trn[state,word]
        mar     *br0+           ; right-shift INDX by one
        xc      1,tc            ; if bit(i) == 1
        adrk    16              ; add 16 to INDX
        sub     #1              ;
        samm    treg2           ; decrement bit pointer
trace:
        ret
```

FACCH Channel Format

The FACCH is a signaling channel in parallel with the speech path used for transmission of control and supervision messages between the base station and the mobile station. The FACCH replaces the user information block (that is, speech data) whenever necessary [1]. An FACCH message block consists of a 48-bit message frame, a 1-bit continuation flag, and a 16-bit CRC. The standard CCITT CRC-16 code is generated for 49 information bits (1 continuation and 48 message) and eight bits of DVCC color code. The FACCH data (48-bit message, 1-bit continuation, 16-bit CRC) is error protected by means of a rate-1/4 convolutional code. The resulting 260-bit frame is interleaved over two consecutive bursts in the same manner as the VSELP speech frame.

The rate-1/4 convolutional encoder has a constraint length of 5. In other words, it operates as a shift register of length 5. Each new bit shifted in results in four parity bits being shifted out of the encoder that are designated P1, P2, P3, and P4. Figure 6 illustrates the encoder shift register.

Figure 6. FACCH Rate-1/4 Convolution Encoder



The 65-bit input frame to the encoder consists of 48 bits of data, a 1-bit continuation flag indicating whether this is the first word of a message, and 16 bits of CRC code. The encoder does not require five explicit tail bits, as was the case with the VSELP rate-1/2 encoder. It treats each input frame as a 65-bit circular buffer. The first five bits in each input frame constitute the initial encoder state (that is, C[4], C[3], C[2], C[1], C[0]). The first output bit quadruple (P1, P2, P3, P4) is generated when the sixth bit is shifted in. After shifting the 65th bit in, bit 0 is input to the encoder, creating the circular buffer. The final encoder state is (C[3], C[2], C[1], C[0], C[64]). Note that after one more shift, the encoder state would return to its initial state. In terms of the corresponding trellis structure, this means that there is always a wraparound from the final encoder state to its initial state.

The FACCH decoder is similar to the VSELP decoder except for the following considerations:

- It decodes rate-1/4 code instead of rate-1/2 code.
- The encoder frame is 65 bits long.
- Each encoder frame is treated as a circular buffer.

The basic Viterbi algorithm in this case remains identical to the VSELP rate-1/2 algorithm. There are two paths entering each state from the previous symbol interval. The decoder selects the lower cost link, based

on its accumulated cost. However, since each output symbol consists of four bits, there are possibly 16 distinct transition costs that need to be updated at every symbol interval. The rest of the algorithm is similar to the speech decoder algorithm except that the frame size is 65 bits instead of 89 bits.

Since the encoder initial state is not previously known in this case, all states are equally likely in the first symbol interval. Hence, all accumulated costs are initialized to 0 at the beginning of each frame. This can result in poor initial performance of the decoder under low signal-to-noise (SNR) conditions. According to Forney [4], the Viterbi decoder output is unreliable until a path history of four or five times the encoder-constraint length is available. Therefore, the first 20 to 25 decoded bits can contain errors. This problem can be alleviated by considering the final encoder state (in the 65th symbol interval) and the initial encoder state (in the first symbol interval) wraparound. Each received frame is treated as a 65-symbol-long circular buffer, and the decoder is fed with a total of 85 symbols (composed of a 65-symbol frame and 20 repeated initial symbols), thereby generating an artificially long path history. Since 20 initial symbols are repeated, a portion of the path history is redundant. Ideally, path history that corresponds to the first 20 symbol intervals and the last 20 symbol intervals should be identical because it corresponds to the same 20 symbols. However, the trellis generated for the last 20 symbol intervals is more reliable because it takes into account the path history of the previous 65 symbols. Accordingly, the path history of the first 20 symbols is pruned. This approach is taken to avoid the uncertainty of the decoder decisions during the first 20 input symbols. After all the symbols are input to the decoder, the best path (of the possible 32 paths) is selected based on least accumulated cost. This path is traced back to yield the output bit sequence.

Code Availability

The associated program files are available from the Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com.*

References

1. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard*, IS-54 Project Number 2215, Electronic Industries Association, December 1989.

2. *TMS320C5x User's Guide*, Texas Instruments, 1993.

3. Pawate, B. I., "Wireless Communications: A Systems Perspective", Texas Instruments, 1992.

4. Forney, G. D., Jr., "The Viterbi Algorithm", *Proceedings of the IEEE*, March 1973.

5. Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asymptomatically Optimum Decoding Algorithm", *IEEE Transactions, Infinity Theory*, April 1967.