Center for Embedded Computer Systems
University of California, Irvine

A Convolutional Encoder and Decoder Model in SpecC

Siwen Sun, Rainer Dömer

Technical Report CECS-08-03
February 27, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{s.sun, doemer}@uci.edu
http://www.cecs.uci.edu/

A Convolutional Encoder and Decoder Model in SpecC

Siwen Sun, Rainer Dömer

Technical Report CECS-08-03
February 27, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{s.sun, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

System level design, also known as embedded system design, has to cope with constantly increasing computational complexity. One solution to address the complexity problem is the modeling of systems at higher levels of abstraction. In this report, we model a convolutional encoder and decoder system and describe it in SpecC, a system level design language (SLDL). This report documents the system hierarchy, the model source code, and our successful simulation results.

Contents

List of Figures

List of Acronyms

SLDL System Level Design Language.

SOC System-on-Chip.

GSM Global System for Mobile communication.

FSM Finite State Machine.

A Convolutional Encoder and Decoder Model in SpecC

Siwen Sun, Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

{s.sun, doemer}@uci.edu
http://www.cecs.uci.edu

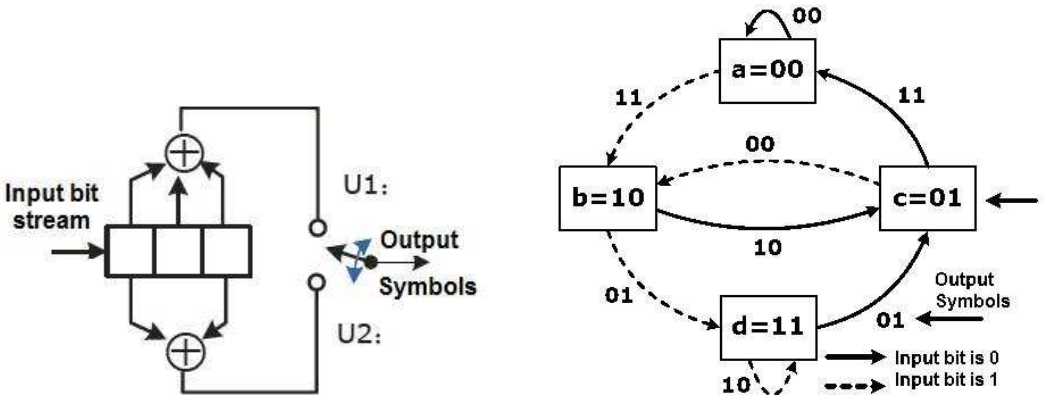
Abstract

System level design, also known as embedded system design, has to cope with constantly increasing computational complexity. One solution to address the complexity problem is the modeling of systems at higher levels of abstraction. In this report, we model a convolutional encoder and decoder system and describe it in SpecC, a system level design language (SLDL). This report documents the system hierarchy, the model source code, and our successful simulation results.

1 Introduction

In this paper, we developed a convolutional encoder and decoder system with SpecC, a system level design language which is intended for specification and design of system-on-chips (SOCs) and embedded systems [1]. Convolutional encoding and decoding is a channel encoder and decoder technique. It is now widely used in Global System for Mobile Communication (GSM) [2].

This paper is organized as follows: first of all, the convolutional encoding and decoding algorithm is introduced in Section II; then, a SpecC based convolutional encoder and decoder system is designed and modeled in Section III; next, the implementation of this system is given in Section IV, and simulation results are also presented in this section; finally, main conclusions of this paper are highlighted in Section V.



(a) Combinational Logic method (b) FSM method

Figure 1: Two descriptions for $(1/2,3)$ Convolutional Coder

2 Convolutional Code Algorithm

The convolutional coder is often used in digital communication systems where the signal to noise ratio is very low, because it is able to achieve error free transmission by adding enough redundancy to the source symbols. "Convolutional" means the current output channel symbols are not only derived from the current input bits, but from a combinatorial logic that is fed by all the bits within a constraint length cycle [3].

Standard convolutional codes $(k/n,L)$ are usually described by two parameters: the code rate k/n and the constraint length L, in which k is the number of bits input into the convolutional encoder and n is the number of channel symbols output by the convolutional encoder. In this paper, we mainly concentrate in $(1/2,3)$ convolutional codes, which means the code rate is $1/2$ and constraint length is 3.

Convolutional encoder can be denoted by kinds of methodologies, such as combinational logic circuit, Finite State Machine (FSM), Trellis diagram, or code generator polynomials. Figure 1 depicts the first two approaches for a standard $(1/2,3)$ convolutional encoder in I-GSM-95 standard. For the convolutional encoder shown in Figure 1, if the input bit stream is 01001101101, the output symbols stream should be 10111101010001010010 00.

The realization of convolutional coder system in embedded system can be described by an FSM in Figure $2(b)$. And the corresponding decoding process is an anti-route of encoding process.

3 Encoder and Decoder System

Stimulus: Generate one/zero bit stream of required length. This task is completed by a random number generator.
Encoder: Accept the input information frame from Stimulus and then performs the specified convolutional encoding and output encoded frame consisting of one/zero channel symbols.
Decoder: First perform hard-decision on input code frames; then build corresponding decoding states and output the decoded information frame.
Encoding State: Child behaviors of Encoder; each state denotes a stage in encoding process. All the encoding states along with corresponding state transition rules compose a FSM.
Decoding State: Child behaviors of Decoder; each state denotes a stage in decoding process. All the decoding states along with corresponding state transition rules compose a FSM.
Compare: Compares the source data output by Stimulus to the data output by the decoder and counts the number of errors.
Monitor: Display the information bit stream from the Stimulus, the encoded frame from the Encoder, as well as the decoded frame from the Decoder.

All the communications between the above design units adopt hand-shaking protocol. Two kinds of specified hand-shaking channels are implemented, data channel and code channel. Data channel is for source information frames and decoded frames transmission. Code channel is used in transmission of encoded frames.

4 Model and Simulation

To simulate a $(1/2, 3)$ convolutional coder system, four encoding states are defined, State00, State01, State10, and State11. These four states compose an FSM in the encoder. Correspondingly in the decoder, four decoding states are used for decoding, DecState00, DecState01, DecState10, and DecState11. Two frame structures are defined, DataMsg and CodeMsg. CodeMsg has a twice length of DataMsg. The Monitor has three child behaviors, connecting with Stimulus, Encoder, and Decoder respectively. The detailed hierarchical structure of the test bench is depicted in Figure 2 as follows.

Then we test this system for single frame and block of frames both. In I-GSM-95 channel encoding, each 20 ms speech frame has 260 bits, 189 out of which are convolutional encoded and the rest bits are encoded with cyclic codec [2]. After rate $1/2$ encoding, the output frame should be 378 bits ($189 * 2$ without protection bits). Then the 20 ms speech data changes into 456 bits after channel encoding and the data rate becomes 22.8 kbps. Before interleaving, the 456 bits are divided into 8 frames, with each frame having 57 bits. Our single frame simulation tests a 189-bit frame and the block of frames simulation tests a block of eight 57-bit frames. Both simulations run fine and the number of decoding errors turns out to be zero.

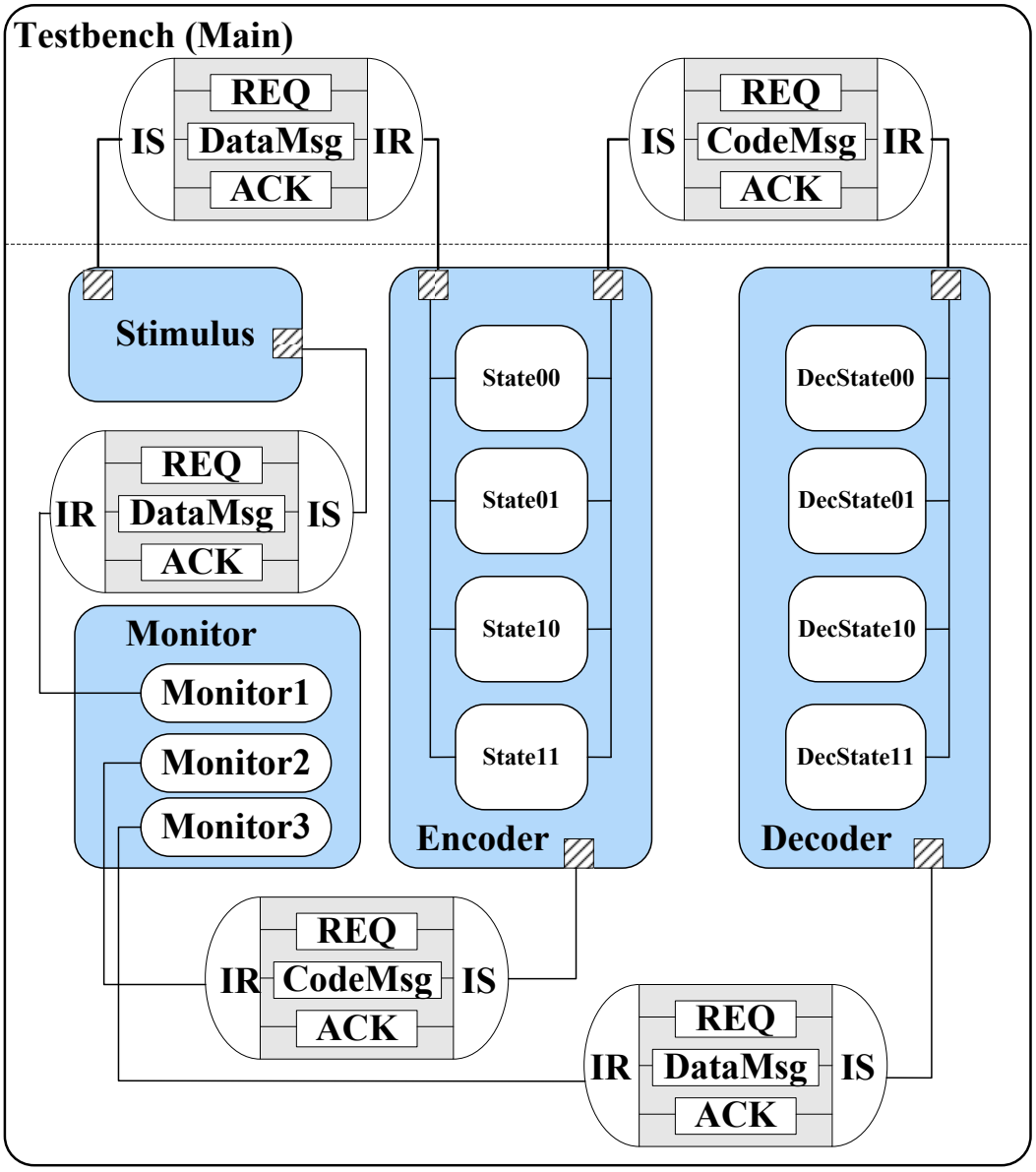


Figure 2: The Hierarchical Structure of Test bench with Child Behaviors and Channels

5 Conclusion

In this report, a $(1/2, 3)$ convolutional encoder and its corresponding decoder are designed, modeled, and implemented with SpecC. The related theory, system organization, and SpecC methodology are given. The program works well and the simulation results are correct. Detailed SpecC program and run log are presented in the Appendix.

References

[1] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, Shuqing Zhao, *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[2] Bernard. Sklar, *Digital Communications: Fundamentals and Applications 2nd Edition*. Prentice Hall PTR, 2001.

[3] Andrew. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," in *IEEE Transactions on Information Theory*, vol. IT-13, pp. 260–269, 1967.

A Appendix

A.1 Simulation results

——-Standard I-GSM-95 189bit Single Frame Test——

The input frame is
010011011010100100110010000110100100001101000110010001010011000010001010001010001000100101111000
001101101011100010000110111110110001001111110001010110100110010000001100111100100001011110011

The encoded frame is
101111010100010100100010111110111101011111110110000110101001011111011000011010100101100110101111
101100111000101111010111000011101100111000101100111000101100111011001110111110000110100111000000
110101000101001000011001110011101100001101010001101010010001011100111011110110101010011100111000
100001010010111101011111101100000000110101111101101001111110110000111000011010011111010111

The decoded frame is
010011011010100100110010000110100100001101000110010001010011000
010001010001010001000100101111000001101101011100010000110111110
110001001111110001010110100110010000001100111100100001011110011

There are 0 bit error in decoded frames

——Standard I-GSM-95 8 frames pre-interleaving——
———57 bit for each frame———

1 frame: 01011110101110001111000110100000110101100111101
There are 0 bit error in decoded frames

2 frame: 01000111110010001100011110001101111101110011010
There are 0 bit error in decoded frames

3 frame: 11010000001100000001111000111101011000100010010
There are 0 bit error in decoded frames

4 frame: 01010110100110010000001100111100100001011110011
There are 0 bit error in decoded frames

5 frame: 11111111010110010101101111000000000001110000011
There are 0 bit error in decoded frames

6 frame: 10100011000101001110011000101000011110010101011
There are 0 bit error in decoded frames

7 frame: 01001010101100001010101001000000101111010111111
There are 0 bit error in decoded frames

8 frame: 10111000111111001111010010111000110111110111000
There are 0 bit error in decoded frames

The total bit error is 0

A.2 Source Code

A.2.1 head.sh

```
/* head.sh
Headfile for convolutional encoder and decoder
according to i-GSM-95 standard
12/13/2007 */

#include<stdio.h>
#include<stdlib.h>
#include<sim.sh>
#ifdef blocksize
#define framelength 47
#else
#define framelength 189
#endif

typedef unsigned bit[framelength] DataFrame;
typedef unsigned bit[framelength*2] CodeFrame;
```

A.2.2 Communication.sc

```
/* Communication.sc
Channels Definition for single frame test
12/13/2007 */

#include "head.sh"

interface SendData
{
        void SendDataFrame(DataFrame);
};
interface ReceiveData
{
        DataFrame ReceiveDataFrame(void);
};
interface SendCode
{
        void SendCodeFrame(CodeFrame);
};
interface ReceiveCode
{
        CodeFrame ReceiveCodeFrame(void);
};//Channel for source information frame transmission

channel DataTrans implements SendData,ReceiveData
{
        event Req;
        DataFrame Data;
        event Ack;

        void SendDataFrame(DataFrame msg)
```

```
        {
                Data=msg;
                notify Req;
                wait Ack;
        }

        DataFrame ReceiveDataFrame(void)
        {
                DataFrame r-msg;
                wait Req;
                r-msg=Data;
                notify Ack;
                return r-msg;
        }
};//Channel for encoded information frame transmission

channel FrameTrans implements SendCode,ReceiveCode
{
        event Req;
        CodeFrame Code;
        event Ack;

        void SendCodeFrame(CodeFrame msg)
        {
                Code=msg;
                notify Req;
                wait Ack;
        }

        CodeFrame ReceiveCodeFrame(void)
        {
                CodeFrame r-msg;
                wait Req;
                r-msg=Code;
                notify Ack;
                return r-msg;
        }
};
```

A.2.3 sim1.sc

```
/* sim1.sc
Definition of the behavior stimulus and monitor for single frame test
12/13/2007 */

#include "head.sh"
import "Communication";

behavior Stimulus(inout DataFrame msg, SendData OutPort, SendData Display)
{
        void main()
        {
                int i=0;
```

```
                srand(4);
                  while(i<framelength)
                  {
                          msg[i]=(rand()%2==0)?0:1;
                          i++;
                  }
                          Display.SendDataFrame(msg);
                          OutPort.SendDataFrame(msg);
        }
};//Monitor1 is for the display of source informaiton from stimulus

behavior Monitor1(inout DataFrame msg, ReceiveData InPort)
{
        void main()
        {
                int i;
                msg=InPort.ReceiveDataFrame();

        }
};//Monitor2 is for the display of encoded informaiton from encoder

behavior Monitor2(inout CodeFrame msg, ReceiveCode InPort)
{
        void main()
        {
                int i;
                msg=InPort.ReceiveCodeFrame();
        }
};//Monitor3 is for the display of decoded informaiton from decoder

behavior Monitor3(inout DataFrame msg, ReceiveData InPort)
{
        void main()
        {
                int i;
                msg=InPort.ReceiveDataFrame();
        }
};

behavior Monitor(inout DataFrame Frame1, inout CodeFrame Frame2,
inout DataFrame Frame3, ReceiveData C1, ReceiveCode C2, ReceiveData C3)
{
    Monitor1 Display1(Frame1,C1);
        Monitor2 Display2(Frame2,C2);
        Monitor3 Display3(Frame3,C3);
        void main()
        {
                int i;
                par{
                        Display1.main();
                        Display2.main();
                        Display3.main();
```

```
                }
                printf("The input frame is");
                for(i=framelength-1;i>=0;i--)
                {
                        printf("%d",(int)Frame1[i]);
                }
                printf("\n\n");
                printf("The encoded frame is");
                for(i=framelength*2-1;i>=0;i--)
                {
                        printf("%d",(int)Frame2[i]);
                }
                printf("\n\n");
                printf("The decoded frame is");
                for(i=framelength-1;i>=0;i--)
                {
                        printf("%d",(int)Frame3[i]);
                }
                printf("\n\n");
        }
};
```

A.2.4 encoder.sc

```
/* convol-encoder.sc
Convolutional encoder for single frame test
12/13/2007 */

#include "head.sh"
import "Communication";

//the follows are four child behaviors for FSM in the encoder
behavior State00(inout int index, in DataFrame datamsg, inout CodeFrame codemsg)
{
     void main()
     {
        if(datamsg[index]==1)
         {
                codemsg[index*2+0]=1;
                codemsg[index*2+1]=1;
         }
         else
         {
                codemsg[index*2+0]=0;
                codemsg[index*2+1]=0;
         }
         index++;
     }
};

behavior State01(inout int index, in DataFrame datamsg, inout CodeFrame codemsg)
{
     void main()
```

```
     {
         if(datamsg[index]==1)
          {
                  codemsg[index*2+0]=0;
                  codemsg[index*2+1]=0;
          }
          else
          {
                  codemsg[index*2+0]=1;
                  codemsg[index*2+1]=1;
          }
             index++;
     }
};

behavior State10(inout int index, in DataFrame datamsg, inout CodeFrame codemsg)
{
    void main()
    {
         if(datamsg[index]==1)
         {
                  codemsg[index*2+0]=1;
                  codemsg[index*2+1]=0;
         }
         else
         {
                  codemsg[index*2+0]=0;
                  codemsg[index*2+1]=1;
         }
         index++;
    }
};

behavior State11(inout int index, in DataFrame datamsg, inout CodeFrame codemsg)
{
    void main()
    {
         if(datamsg[index]==1)
         {
                  codemsg[index*2+0]=0;
                  codemsg[index*2+1]=1;
         }
         else
         {
                  codemsg[index*2+0]=1;
                  codemsg[index*2+1]=0;
         }
         index++;
    }
};

behavior encoder(DataFrame datamsg, CodeFrame codemsg, ReceiveData InPort,
```

```
SendCode Display, SendCode OutPort)
{
        int index=0;
    State00 S-00(index, datamsg, codemsg);
        State10 S-10(index, datamsg, codemsg);
        State01 S-01(index, datamsg, codemsg);
        State11 S-11(index, datamsg, codemsg);
        void main()
        {
                datamsg=InPort.ReceiveDataFrame();
                fsm
                {
            S-00:{ if (index>framelength) break;
                                if (datamsg[index-1]==0) goto S-00;
                        if (datamsg[index-1]==1) goto S-10;}

            S-01:{ if (index>framelength) break;
                                if (datamsg[index-1]==0) goto S-00;
                        if (datamsg[index-1]==1) goto S-10;}

            S-10:{ if (index>framelength) break;
                                if (datamsg[index-1]==0) goto S-01;
                        if (datamsg[index-1]==1) goto S-11;}

            S-11:{ if (index>framelength) break;
                                if (datamsg[index-1]==0) goto S-01;
                        if (datamsg[index-1]==1) goto S-11;}
                }
                OutPort.SendCodeFrame(codemsg);
                Display.SendCodeFrame(codemsg);
        }
};
```

A.2.5 decoder.sc

```
/* decoder.sc
headerbi decoder for single frame test
12/13/2007 */

#include "head.sh"
import "Communication";

//the following are four decoded state in the FSM of the decoder
behavior DecState00(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
     void main()
     {
         index++;
                 if(codemsg[index*2]==0&&codemsg[index*2+1]==0)
                         datamsg[index]=0;
                 else
                         datamsg[index]=1;
         }
```

```
};

behavior DecState01(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
      void main()
      {
          index++;
                  if(codemsg[index*2]==1&&codemsg[index*2+1]==1)
                          datamsg[index]=0;
                  else
                          datamsg[index]=1;
          }
};

behavior DecState10(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
      void main()
      {
          index++;
                  if(codemsg[index*2]==0&&codemsg[index*2+1]==1)
                          datamsg[index]=0;
                  else
                          datamsg[index]=1;
          }
};

behavior DecState11(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
      void main()
      {
          index++;
                  if(codemsg[index*2]==1&&codemsg[index*2+1]==0)
                          datamsg[index]=0;
                  else
                          datamsg[index]=1;
          }
};
//Count the error in the decoded frame
behavior CalError(in DataFrame codemsg, in DataFrame decodemsg)
{
         void main()
         {
                  int i,count=0;
                  for(i=0;i<framelength;i++)
                  {
                          if(codemsg[i]!=decodemsg[i])
                                  count++;
                  }
                  printf("There are %d bit error in decoded frames\n",count);
         }
};
```

```
behavior decoder(CodeFrame codemsg, DataFrame datamsg, ReceiveCode InPort, SendData Display)
{
        int index=-1;
    DecState00 DecS-00(index, codemsg, datamsg);
        DecState10 DecS-10(index, codemsg, datamsg);
        DecState01 DecS-01(index, codemsg, datamsg);
        DecState11 DecS-11(index, codemsg, datamsg);
        void main()
        {
                codemsg=InPort.ReceiveCodeFrame();
                fsm
                {
            DecS-00:{ if (index>framelength) break;
                        if (codemsg[index*2]==0&&codemsg[index*2+1]==0)
                                                goto DecS-00;

                        if (codemsg[index*2]==1&&codemsg[index*2+1]==1)
                                                goto DecS-10;
                    }

            DecS-01:{ if (index>framelength) break;
                        if (codemsg[index*2]==0&&codemsg[index*2+1]==0)
                                                goto DecS-10;

                        if (codemsg[index*2]==1&&codemsg[index*2+1]==1)
                                                goto DecS-00;
                    }

            DecS-10:{ if (index>framelength) break;
                        if (codemsg[index*2]==1&&codemsg[index*2+1]==0)
                                                goto DecS-11;
                        if (codemsg[index*2]==0&&codemsg[index*2+1]==1)
                                                goto DecS-01;
                    }
            DecS-11:{ if (index>framelength) break;
                        if (codemsg[index*2]==0&&codemsg[index*2+1]==1)
                                                goto DecS-11;
                        if (codemsg[index*2]==1&&codemsg[index*2+1]==0)
                                                goto DecS-01;
                    }
             }
                Display.SendDataFrame(datamsg);
        }
};
```

A.2.6 single.sc

```
/* single.sc
testbench for single frame test
12/13/2007 */

#include "head.sh"
import "Communication";
```

```
import "sim1";
import "encoder";
import "decoder";

behavior Main()
{
        DataFrame SourceInfo;
        CodeFrame CodeInfo;
        DataFrame DecodeInfo;
        DataTrans DataChannel1, DataChannel2, DataChannel3;
        FrameTrans CodeChannel1, CodeChannel2;
        Monitor Display(SourceInfo,CodeInfo,DecodeInfo,DataChannel1,CodeChannel1,DataChannel3);
        Stimulus DataInfo(SourceInfo, DataChannel2, DataChannel1);
        encoder Encoder(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
        decoder Decoder(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
        CalError Compare(SourceInfo,DecodeInfo);
        int main(void)
        {
              printf("\n-------Standard I-GSM-95 189 bit Single Frame Test-------\n\n");
                par{
                    DataInfo.main();
                    Encoder.main();
                    Decoder.main();
                    Display.main();
                }
                      Compare.main();
                }
};
```

A.2.7 Communication2.sc

```
/* Communication2.sc
Channels Definition for block of frames test
12/13/2007 */

#define blocksize 8
#include "head.sh"

interface SendData
{
        void SendDataFrame(DataFrame);
};
interface ReceiveData
{
        DataFrame ReceiveDataFrame(void);
};
interface SendCode
{
        void SendCodeFrame(CodeFrame);
};
interface ReceiveCode
{
        CodeFrame ReceiveCodeFrame(void);
```

```
};//Channel for source information frame transmission

channel DataTrans implements SendData,ReceiveData
{
        event Req;
        DataFrame Data;
        event Ack;

        void SendDataFrame(DataFrame msg)
        {
                Data=msg;
                notify Req;
                wait Ack;
        }

        DataFrame ReceiveDataFrame(void)
        {
                DataFrame r-msg;
                wait Req;
                r-msg=Data;
                notify Ack;
                return r-msg;
        }
};//Channel for encoded information frame transmission

channel FrameTrans implements SendCode,ReceiveCode
{
        event Req;
        CodeFrame Code;
        event Ack;

        void SendCodeFrame(CodeFrame msg)
        {
                Code=msg;
                notify Req;
                wait Ack;
        }

        CodeFrame ReceiveCodeFrame(void)
        {
                CodeFrame r-msg;
                wait Req;
                r-msg=Code;
                notify Ack;
                return r-msg;
        }
};
```

A.2.8 sim2.sc

```
/* sim2.sc
Definition of the behavior stimulus and monitor for block of frames test
12/13/2007 */
```

```
#define blocksize 8
#include "head.sh"
import "Communication2";

behavior Stimulus(inout DataFrame msg, SendData OutPort, SendData Display,in int seed)
{
        void main()
        {
                int i=0;
        srand(seed);
                while(i<framelength)
                {
            msg[i]=(rand()%2==0)?0:1;
            i++;
                }
                Display.SendDataFrame(msg);
        OutPort.SendDataFrame(msg);
        }
};

behavior Monitor1(inout DataFrame msg, ReceiveData InPort)
{
        void main()
        {
               int i;
               msg=InPort.ReceiveDataFrame();
        }
};

behavior Monitor2(inout CodeFrame msg, ReceiveCode InPort)
{
        void main()
        {
               int i;
               msg=InPort.ReceiveCodeFrame();
        }
};

behavior Monitor3(inout DataFrame msg, ReceiveData InPort)
{
        void main()
        {
               int i;
               msg=InPort.ReceiveDataFrame();
        }
};

behavior Monitor(inout DataFrame Frame1, inout CodeFrame Frame2, inout DataFrame Frame3,
ReceiveData C1, ReceiveCode C2, ReceiveData C3, inout int index)
{
    Monitor1 Display1(Frame1,C1);
```

```
        Monitor2  Display2(Frame2,C2);
        Monitor3  Display3(Frame3,C3);
        void main()
        {
                int i;
           par{
                                Display1.main();
                                Display2.main();
                                Display3.main();
                        }
           printf("%d frame: ", index);
           for(i=framelength-1;i>=0;i--)
           {
                 printf("%d",(int)Frame1[i]);
           }
           printf("\n");
        }
};
```

A.2.9 decoder2.sc

```
/* decoder2.sc
headerbi decoder for block of frames test
12/13/2007 */

#define blocksize 8
#include "head.sh"
import "Communication2";

behavior DecState00(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
     void main()
     {
                index++;
                if(codemsg[index*2]==0&&codemsg[index*2+1]==0)
                        datamsg[index]=0;
                else
                        datamsg[index]=1;
         }
};

behavior DecState01(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
     void main()
     {
                index++;
                if(codemsg[index*2]==1&&codemsg[index*2+1]==1)
                        datamsg[index]=0;
                else
                        datamsg[index]=1;
         }
};
```

```
behavior DecState10(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
      void main()
      {
                  index++;
                  if(codemsg[index*2]==0&&codemsg[index*2+1]==1)
                        datamsg[index]=0;
                  else
                        datamsg[index]=1;
          }
};

behavior DecState11(inout int index, in CodeFrame codemsg, inout DataFrame datamsg)
{
      void main()
      {
                  index++;
                  if(codemsg[index*2]==1&&codemsg[index*2+1]==0)
                        datamsg[index]=0;
                  else
                        datamsg[index]=1;
          }
};

behavior CalError(in DataFrame codemsg, in DataFrame decodemsg, inout int count)
{
         void main()
         {
                  int i;
                  count=0;
                  for(i=0;i<framelength;i++)
                  {
                           if(codemsg[i]!=decodemsg[i])
                                    count++;
                  }
                  printf("There are %d bit error in decoded frames\n\n",count);
         }
};

behavior decoder(CodeFrame codemsg, DataFrame datamsg, ReceiveCode InPort,
SendData Display)
{
         int index=-1;
     DecState00 DecS-00(index, codemsg, datamsg);
         DecState10 DecS-10(index, codemsg, datamsg);
         DecState01 DecS-01(index, codemsg, datamsg);
         DecState11 DecS-11(index, codemsg, datamsg);
         void main()
         {
                  codemsg=InPort.ReceiveCodeFrame();
                  fsm
                  {
```

```
            DecS-00:{ if (index>framelength) break;
                      if (codemsg[index*2]==0&&codemsg[index*2+1]==0)
                                                goto DecS-00;
                      if (codemsg[index*2]==1&&codemsg[index*2+1]==1)
                                                goto DecS-10;
                            }
            DecS-01:{ if (index>framelength) break;
                      if (codemsg[index*2]==0&&codemsg[index*2+1]==0)
                                                goto DecS-10;
                      if (codemsg[index*2]==1&&codemsg[index*2+1]==1)
                                                goto DecS-00;
                      }
            DecS-10:{ if (index>framelength) break;
                      if (codemsg[index*2]==1&&codemsg[index*2+1]==0)
                                                goto DecS-11;
                      if (codemsg[index*2]==0&&codemsg[index*2+1]==1)
                                                goto DecS-01;
                    }
            DecS-11:{ if (index>framelength) break;
                      if (codemsg[index*2]==0&&codemsg[index*2+1]==1)
                                                goto DecS-11;
                       if (codemsg[index*2]==1&&codemsg[index*2+1]==0)
                                                goto DecS-01;
                      }
             }
                Display.SendDataFrame(datamsg);
        }
};
```

A.2.10 block.sc

```
/* block.sc
testbench for block of frames test
12/13/2007 */

#define blocksize 8
#include "head.sh"
import "Communication2";
import "sim2";
import "encoder2";
import "decoder2";

behavior Main()
{
        int seed,error,total-error;
        DataFrame SourceInfo;
        CodeFrame CodeInfo;
        DataFrame DecodeInfo;
        DataTrans DataChannel1, DataChannel2, DataChannel3;
        FrameTrans CodeChannel1, CodeChannel2;

        Monitor Display(SourceInfo,CodeInfo,DecodeInfo,DataChannel1,CodeChannel1,
        DataChannel3, seed);
```

```
    CalError Compare(SourceInfo,DecodeInfo,error);
Stimulus DataInfo(SourceInfo, DataChannel2, DataChannel1,seed);
encoder Encoder1(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder1(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder2(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder2(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder3(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder3(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder4(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder4(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder5(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder5(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder6(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder6(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder7(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder7(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);
encoder Encoder8(SourceInfo,CodeInfo,DataChannel2, CodeChannel1, CodeChannel2);
decoder Decoder8(CodeInfo,DecodeInfo,CodeChannel2,DataChannel3);

int main(void)
{
      int i=1;
      total-error=0;
      seed=0;
      printf("\n-------Standard I-GSM-95 8 frames pre-interleaving -------\n");
      printf("--------57 bit for each frame---------\n\n");
         seed++;
          par{
               DataInfo.main();
               Encoder1.main();
               Decoder1.main();
               Display.main();
              }
          Compare.main();
          total-error+=error;
          seed++;
          par{
               DataInfo.main();
               Encoder2.main();
               Decoder2.main();
               Display.main();
              }
          Compare.main();
          total-error+=error;
          seed++;
          par{
               DataInfo.main();
               Encoder3.main();
               Decoder3.main();
               Display.main();
              }
          Compare.main();
```

```
                total_error+=error;
                seed++;
                par{
                    DataInfo.main();
                    Encoder4.main();
                    Decoder4.main();
                    Display.main();
                   }
                Compare.main();
                total_error+=error;
                seed++;
                par{
                    DataInfo.main();
                    Encoder5.main();
                    Decoder5.main();
                    Display.main();
                   }
                Compare.main();
                total_error+=error;
                seed++;
                par{
                    DataInfo.main();
                    Encoder6.main();
                    Decoder6.main();
                    Display.main();
                   }
                Compare.main();
                total_error+=error;
                seed++;
                par{
                    DataInfo.main();
                    Encoder7.main();
                    Decoder7.main();
                    Display.main();
                   }
                Compare.main();
                total_error+=error;
                seed++;
                par{
                    DataInfo.main();
                    Encoder8.main();
                    Decoder8.main();
                    Display.main();
                   }
                Compare.main();
                total_error+=error;
            printf("The total bit error is %d\n\n",total_error);
        }
};
```

A.2.11 Makefile

```
#Makefile for convolutional encoder and headerbi decoder
```

```
#last edited: 12/13/2007

SCC      = scc
SCCOPT   = -vvv -ww -sl
SCC2SIR  = -sc2sir $(SCCOPT)
ALL      = single block

all: $(ALL)

single: head.sh single.sc encoder.sir decoder.sir sim1.sir Communication.sir
        $(SCC) single $(SCCOPT)

encoder.sir: head.sh encoder.sc Communication.sir
        $(SCC) encoder $(SCC2SIR)

Communication.sir: head.sh Communication.sc
        $(SCC) Communication $(SCC2SIR)

sim1.sir: head.sh sim1.sc Communication.sir
        $(SCC) sim1 $(SCC2SIR)

decoder.sir: head.sh decoder.sc Communication.sir
        $(SCC) decoder $(SCC2SIR)

block: head.sh block.sc encoder2.sir decoder2.sir sim2.sir Communication2.sir
        $(SCC) block $(SCCOPT)

encoder2.sir: head.sh encoder2.sc Communication2.sir
        $(SCC) encoder2 $(SCC2SIR)

Communication2.sir: head.sh Communication2.sc
        $(SCC) Communication2 $(SCC2SIR)

sim2.sir: head.sh sim2.sc Communication2.sir
        $(SCC) sim2 $(SCC2SIR)

decoder2.sir: head.sh decoder2.sc Communication2.sir
        $(SCC) decoder2 $(SCC2SIR)

clean:
        -rm -f *~ *.o *.cc *.h encoder
        -rm -f *.si
        -rm -f *.sir
        -rm -f *.out

test:
        @echo "./single"
        @echo "./block"

# EOF
```