Coduri convoluționale. Decodoare Viterbi

Codurile convoluționale diferă de codurile bloc atât prin faptul că există memorie pentru codor, cât şi prin dependența celor *n* ieşiri, la fiecare unitate de timp, nu numai de cele *k* intrări, ci şi de cele *m* blocuri anterioare de intrare.

Un cod convoluțional *(n,k,m)* poate fi implementat cu un circuit secvențial care are *n* ieşiri liniare, *k* intrări şi *m* intrări de memorie. Uzual, *n* şi *k* sunt valori întregi cu $k < n$, dar valoarea lui *m* trebuie să fie mai mare pentru a avea probabilitate de eroare mică. În cazul în care k=1, secvența de informație nu mai este divizată în blocuri şi poate fi procesată în mod continuu.

Codurile convoluționale au fost introduse în 1955 de către Elias ca o alternativă a codurilor bloc. La scurt timp după aceea, Wozencraft a propus *decodarea secvențială* ca fiind o metodă eficientă pentru codurile convoluționale.

În 1963, Massey a propus o metodă de decodare mai puțin eficientă, dar mai simplu de implementat , numită *decodarea cu prag.* Aceasta a dat naştere numeroaselor aplicații ale codurilor convoluționale în cadrul transmisiilor digitale prin cablu sau unde radio.

În 1967, Viterbi a propus o metodă de decodare de plauzibilitate maximă, uşor de implementat pentru codurile cu memorie mică. Această schemă, denumită *decodorul Viterbi*, împreună cu versiunea îmbunătățită a decodării secvențiale, au lărgit şi mai mult domeniul de aplicație al codurilor convoluționale spre comunicațiile prin satelit, la începutul anilor 1970.

Codarea codurilor convoluționale

Codorul pentru codul binar (2,1,3) este prezentat în figura 3.1. Se observă că acesta are un registru cu *m=3* deplasări succesive, *n=2* sumatoare modulo 2 şi un multiplexor care serializează ieşirile codorului. Cele două sumatoare pot fi implementate în porți XOR. Deoarece adunarea modulo 2 este o operație liniară, codorul este un registru de deplasare cu reacție pozitivă. Astfel, toate codoarele convoluționale pot fi implementate cu registre de acest tip.

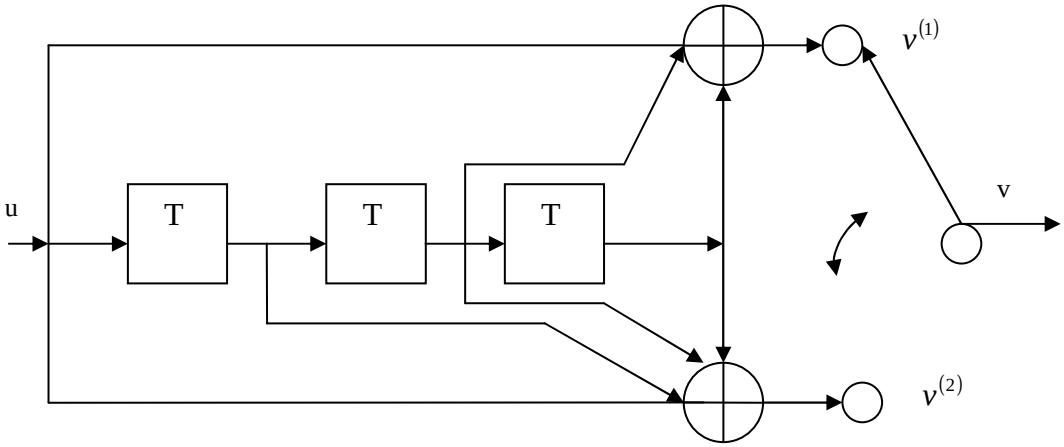


Fig. 3.1. Codorul convoluțional binar (2,1,3).

Secvența de informație $u=(u_1,u_2,\ldots)$ intră în codor pas cu pas, bit cu bit. Dacă codorul este un sistem linar, cele două secvențe de ieşire $v^{(1)}=(v_0^{(1)},v_1^{(1)},v_2^{(1)},\ldots)$ şi $v^{(2)}=(v_0^{(2)},v_1^{(2)},v_2^{(2)},\ldots)$ pot fi obținute prin convoluția secvenței de intrare *u* şi a celor două răspunsuri la impuls ale codorului. Răspunsurile la impuls sunt obținute prin introducerea secvenței de informație *u=(100 ...)* şi prin observarea celor două secvențe de ieşire.

Răspunsurile la impuls ale codorului au pentru cele *m* registre de memorie cel mult *m+1* unități de deplasare, putând fi scrise:

$$g^{(1)}=(g_0^{(1)},g_1^{(1)},\ldots,g_m^{(1)});$$
$$g^{(2)}=(g_0^{(2)},g_1^{(2)},\ldots,g_m^{(2)}).$$

Pentru codorul din figura 3.1,

$$g^{(1)}=(1011),$$
$$g^{(2)}=(1111).$$

Cele două răspunsuri la impuls $g^{(1)}$ şi $g^{(2)}$ se numesc *secvențe generatoare* ale codului. Se pot scrie în acest mod ecuațiile de codare:

$$v^{(1)}=u*g^{(1)},$$
$$v^{(2)}=u*g^{(2)},$$

unde * indică convoluția discretă şi că toate operațiile sunt modulo 2. Operația de convoluție presupune că pentru orice $l\geq 0$,

$$v_l^{(j)}=\sum_{i=0}^{m}u_{l-i}g_i^{(j)}=u_lg_0^{(j)}+u_{l-1}g_1^{(j)}+\cdots+u_{l-m}g_m^{(j)},\quad j=1,2,...$$

unde $u_{l-i}\overset{\Delta}{=}0$ pentru orice l < i. Astfel pentru codorul din figura 3.1,

$$v_l^{(1)}=u_l+u_{l-2}+u_{l-3}$$

$$v_l^{(2)} = u_l + u_{l-1} + u_{l-2} + u_{l-3}$$

pot fi uşor verificate prin observarea directă a circuitului de codare. După codare cele două secvențe de ieşire sunt multiplexate într-o singură secvență, numită *cuvânt de cod*, pentru a fi transmisă prin canal. Cuvântul de cod este exprimat prin:

$$v = \left(v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_1^{(2)}, \ldots\right).$$

Un cod convoluțional generează *n* biți de codare pentru fiecare *k* biți de informație, raportul *R=k/n* numindu-se *rata codului*. Se observă ca pentru o secvență de informație de lungime finită *k·L*, cuvântul de cod corespunzător are lungimea *n(L+m)*, unde cele *n·m* ieşiri finale sunt generate după ce ultimul bloc de informație nenul a intrat în codor. Privind codul convoluțional ca pe un cod bloc liniar cu matricea generatoare G, rata codului bloc este dată de kL/n(L+m), raportul numărului biților de informație şi al lungimii cuvântului de cod.

Dacă L >> m, atunci $L/(L+m) \approx 1$, deci rata codului bloc şi a codului convoluțional sunt aproape egale. Acesta reprezintă modul normal de lucru al codului convoluțional şi de acum înainte nu se mai face distincție între rata codului convoluțional şi rata acestui cod văzut ca un cod bloc. Dacă L are o valoare mică, se va putea prezenta rata codului ca fiind un raport simplificat

$$\frac{k/n - kL/n(L+m)}{k/n} = \frac{m}{L+m},$$

numit *rata fracționară de pierderi.* Pentru a avea rata fracționară de pierderi cât mai mică (adică 0), *L* trebuie propus a fi mult mai mare decât *m*.

Decodarea codurilor convoluționale

Codurile convoluționale suportă o multitudine de metode de decodare, printre care: decodarea cu prag, decodarea de plauzibilitate maximă, decodarea secvențială (algoritmii Stack şi Fano), decodarea cu logică majoritară (feedback), decodarea pentru pachete de erori sau pentru combinații de pachete de erori şi erori aleatoare etc.

Aplicație coduri convoluționale

Următoarea aplicație utilizează codurile convoluționale.

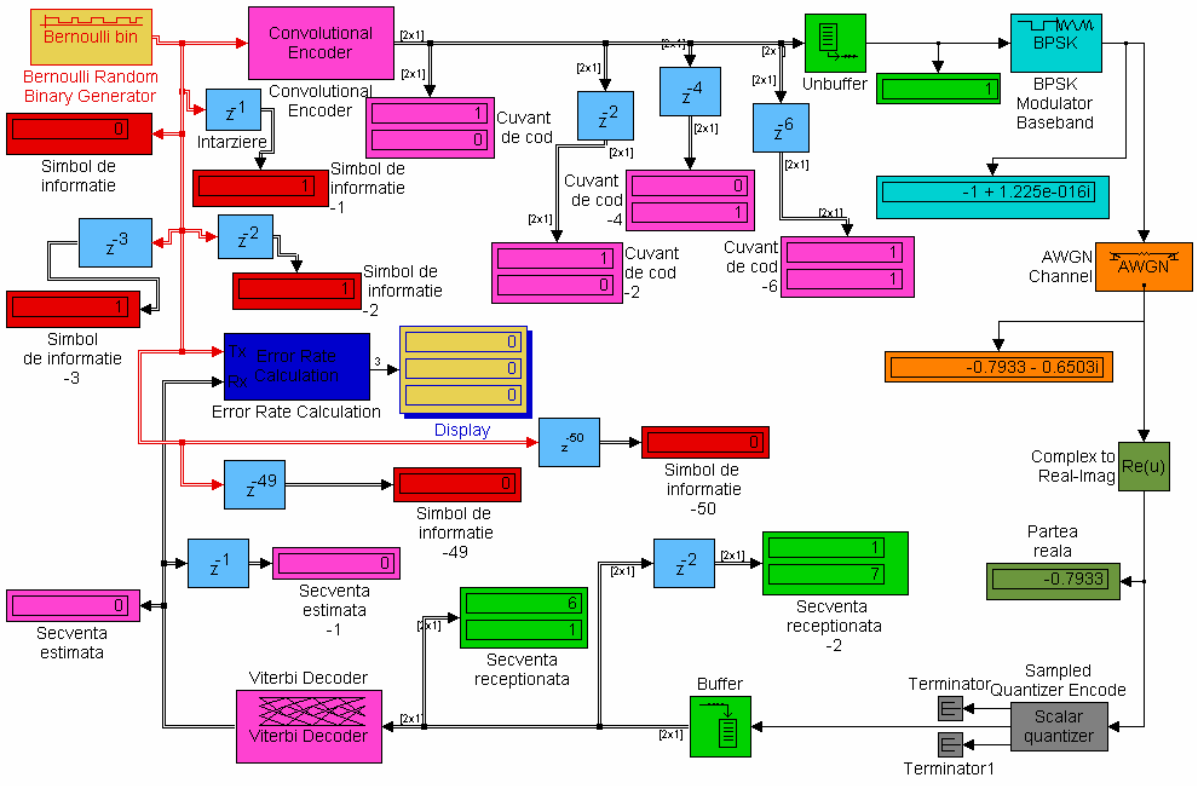


Fig. 3.2

Simularea începe prin generarea unor mesaje binare aleatoare. Se codează mesajul printr-un cod convoluțional, apoi se modulează utilizând modulația BPSK (Binary Phase Shift Keying) şi se adaugă zgomot alb Gaussian pentru a simula un canal cu zgomot. Apoi se decodează codul convoluțional încercând să se corecteze cât mai multe erori introduse de zgomot. În final se compară informația decodată cu mesajul original pentru a prelucra şi afişa rata de eroare.

Blocurile utilizate în model sunt:

- **Bernoulli Random Binary Generator**
- **Convolutional Encoder**
- **Unbuffer**
- **BPSK Modulator Baseband**
- **AWGN Channel**
- **Complex to Real-Imag**
- **Sampled Quantizer Encode**
- **Buffer**
- **Viterbi Decoder**
- **Error Rate Calculation**

Blocul **Bernoulli Random Binary Generator** produce secvența de informație care urmează să fie codată. Deoarece parametrul ‘Sample time’ este 1 secundă, blocul generează un număr binar în fiecare secundă. Parametrul ‘Probability of a zero’ este 0.5, însemnând că se generează biții astfel încât 0 şi 1 au aceeaşi probabilitate de apariție.

Parametrul ‘Initial seed’ inițializează generatorul de numere aleatoare. Dacă se modifică acest parametru va fi generată altă secvență aleatoare. Deoarece parametrul ‘Frame-based outputs’ este marcat înseamnă că se introduc trenuri de impulsuri de semnal.

Blocul **Convolutional Encoder** primeşte secvența de informație de la blocul **Bernoulli Random Binary Generator** şi o transformă în cuvântul de cod. În timp ce secvența de informație este un şir de biți, secvența de ieşire este un vector binar de lungime doi.

Codorul pentru codul (2,1,6) utilizat este reprezentat în fig. 3.3.

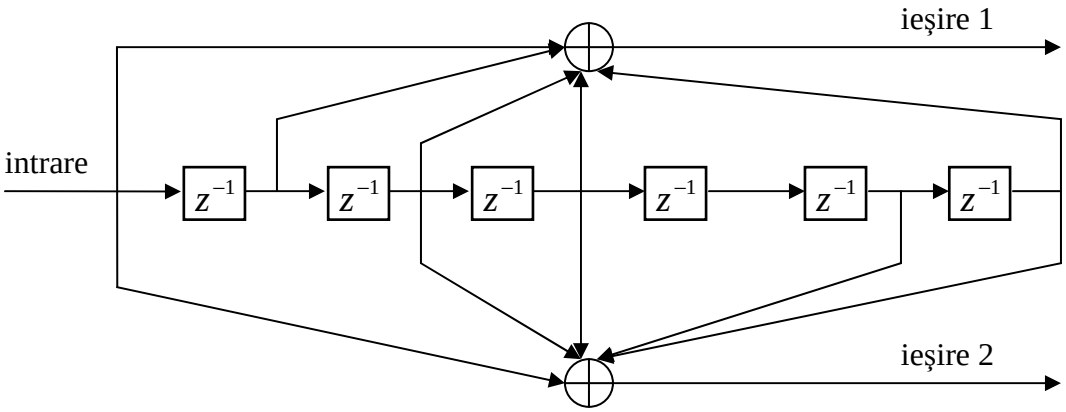


Fig. 3.3

Codorul are n=2 sumatoare modulo 2, un multiplexor şi un registru de memorie cu m=6 deplasări succesive. Deoarece sunt 6 deplasări succesive, ieşirea depinde de 7 valori de intrare (incluzînd-o şi pe cea curentă). Rata codului este ½ deoarece avem o intrare şi două ieşiri. Legătura între registrul de memorie şi sumatoare este dată de generatorul codului care în acest caz este dat de perechea [171 133] (informația de la prima ieşire din codor este dată de următoarea secvență [1111001] care este echivalentă cu numărul 171, iar informația de la a doua ieşire este dată de secvența [1011011] care este echivalentă cu numărul 133).

Parametrul ‚Trellis structure**’** este în acest caz dat de structura poly2trellis(7, [171 133]).

Ieşirea codorului va fi o matrice 2x1 de numere binare. Primul element din matrice arată valoarea de la prima ieşire iar al doilea element arată valoarea de la a doua ieşire.

În modelul din fig. 3.2 simularea se realizează pentru un interval de timp de la 0 la 3, generându-se astfel 4 numere binare.

Conform schemei de codare avem:

- se generează **1** care este codat ca [**11**] deoarece nu mai avem alte numere memorate în registrul de memorie;
- se generează **1** care se codează ca [**01**] deoarece: 1+1=0 (ieşirea 1),
  1+0=1 (ieşirea 2);
- se generează **1** care se codează ca [**10**] deoarece: 1+1+1=1 (ieşirea 1),
  1+0+1=0 (ieşirea 2);
- se generează **0** care se codează ca [**10**] deoarece: 0+1+1+1=1 (ieşirea 1),
  0+0+1+1=0 (ieşirea 2).

Dacă generăm 5 numere al cincilea ar fi **1** care se codează [**11**] deoarece 1+0+1+1+0=1 (ieşirea 1), 1+0+1+1+0=1 (ieşirea 2).

După ce codorul convoluțional codează numerele, acestea sunt modulate. Deoarece cuvintele de cod sunt vectori bidimensionali se introduce blocul **Unbuffer** care transformă vectorul bidimensional într-un semnal scalar. Intrarea blocului **Unbuffer** este un vector bidimensional generat la interval de o secundă, iar ieşirea este un scalar generat la jumătate de secundă.

Blocul **BPSK Modulator Baseband** modulează semnalul scalar de date având la ieşire un semnal complex.

După modulație datele sunt pregătite de transmitere.

Blocul **AWGN Channel** simulează un canal cu zgomot adăugând zgomot alb Gaussian la şirul de date. Se utilizează o distribuție Gaussiană determinată de parametrii: 'Es/No', care este - 1 decibel; 'Input signal power', care este 1 watt deoarece modulația BPSK produce valori de -1 şi 1; 'Symbol period', care este 0.5 secunde. Parametrul 'Initial seed' este 123456 (dacă se modifică se va genera altă secvență de numere).

Datele de la ieşirea acestui bloc sunt numere complexe apropiate de -1 şi 1. Ele trebuiesc transformate astfel încât să corespundă cu semnalul de intrare de la decodorul Viterbi.

Decodorul Viterbi este configurat să proceseze valori întregi în intervalul [0,7].

Blocul **Complex to Real-Imag** transformă datele de la intrare într-un semnal real înlăturând partea imaginară (care este aproximativ egală cu zero).

Blocul **Sampled Quantizer Encode** transformă semnalul real într-un întreg în intervalul [0,7], pentru a-l pregăti pentru algoritmul cu decizii soft.

Parametrul 'Quantization codebook' este un vector de forma [76543210].

Parametrul 'Quantization partition' este [-.75-.5-.250.25.5.75] deoarece intrarea în acest bloc este apropiată de valorile -1 şi 1. Primul grup este format din numerele mai mici sau egale cu -0.75 ; al doilea grup este format din numerele între -0.75 şi -0.5; al treilea grup este format din numerele între -0.5 şi -0.25; şi aşa mai departe până la al optălea grup care este format din numerele mai mari decât 0.75. O valoare de -1 de la intrare este transformată în 7 la ieşire iar o intrare de 1 este transformată în 0 la ieşire.

Deoarece semnalul de la ieşirea blocului este un scalar, acesta este introdus într-un bloc **Buffer** care să-l transforme într-un vector bidimensional. Astfel ieşirea din acest bloc este corespunzătoare ca intrare pentru blocul **Viterbi Decoder**. Ieşirea blocului este bidimensională deoarece parametrul 'Output buffer size' este 2. Blocul combină primele două eşantioane într-un singur vector bidimensional de ieşire, apoi al treilea cu al patrulea eşantion în al doilea vector de ieşire, şi aşa mai departe. Perioada ieşirilor blocului este de două ori mai mare decât perioada intrărilor. La aceste bloc va apare o întârziere de o secundă.

Blocul **Viterbi Decoder** poate acum decoda datele de la intrare. Parametrul 'Trellis structure' defineşte decodorul şi este identic cu cel de la codorul convoluțional. Blocul utilizează decizia soft cu 2^3 valori de intrare diferite deoarece parametrul 'Decision type' este 'Soft Decision' iar parametrul 'Number of soft decision bits' este 3. Blocul

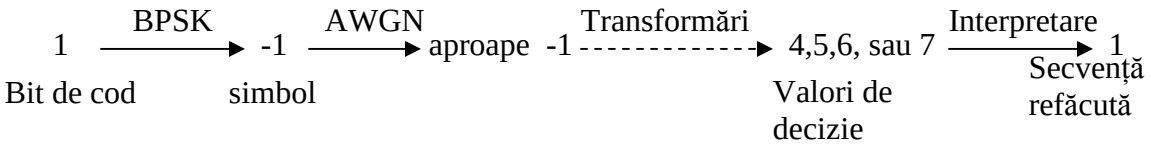
interpretează 0 ca fiind decizie sigură că bitul de cod este 0 , iar 2^3-1 ca fiind decizie sigură că bitul de cod este 1. Valorile dintre 0 şi 7 sunt decizii mai puțin sigure.

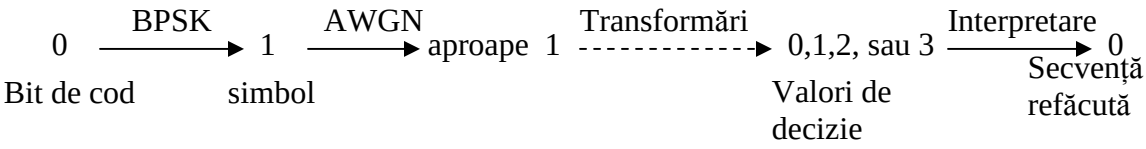
Tabelul următor arată interpretarea celor opt valori de intrare.

Valoarea de decizie	Interpretare
0	Decizie sigură 0
1	Decizie mai puțin sigură 0
2	Decizie şi mai puțin sigură 0
3	Decizie nesigură 0
4	Decizie nesigură 1
5	Decizie şi mai puțin sigură 1
6	Decizie mai puțin sigură 1
7	Decizie sigură 1

Fig. 3.4

În continuare voi arăta modul de circulație al unui bit între blocul **Unbuffer** şi blocul **Viterbi Decoder.**





Parametru 'Traceback Depth' reprezintă întârzierea de decodare. De obicei întârzierea se alege de cinci sau şase ori mai mare decât lungimea şirului, care este şapte. Unele implementări hardware au opțiune de 48 sau 92. Aleg 48 deoarece este mai aproape de 35 sau 42.

Se observă că atunci când se generează 1(pasul 1) vom avea la ieşirea din codor [11] iar la intrarea în decodor [00] deoarece apare o întârziere de o secundă dată de blocul **Buffer**; când se generează 1 (pasul 2) vom avea la ieşirea din codor [01] iar la intrarea în decodor [57] care va fi interpretat ca [11] de către acesta şi corespunde secvenței generate la pasul 1; când se generează 1 (pasul 3) vom avea la ieşirea din codor [10] iar la intrarea în decodor [17] care va fi interpretat ca [01] de către acesta şi corespunde secvenței generate la pasul 2; când se generează 0 (pasul 4) vom avea la ieşirea din codor [10] iar la intrarea în decodor [61] care va fi interpretat ca [10] de către acesta şi corespunde secvenței generate la pasul 3; ş.a.m.d.

Blocul **Error Rate Calculation** compară secvența de informație de la blocul **Bernoulli Random Binary Generator** cu secvența estimată de la blocul **Viterbi Decoder** şi produce trei vectori care reprezintă: rata de eroare; numărul total de erori;

numărul total de comparații. Deoarece parametrul ‘Output data’ este ‘Port’, blocul trimite datele de ieşire la un **Display.**

Parametrul ‘Receive delay’ spune blocului cu ce element de la intrare să realizeze comparația. În acest caz valoarea parametrului este 49, fiind cu unu mai mare decât valoarea parametrului ‘Traceback depth’ care este 48 (blocul **Viterbi Decoder)**. Întârzierea de 1 este dată de blocul **Buffer** (el colectează două eşantioane scalare pentru a scoate un vector bidimensional).

Blocul **Display** primeşte date de la blocul **Error Rate Calculation** pe care le afişează.

Blocul **Terminator** este ataşat porturilor de ieşire ale căror semnale nu sunt utilizate.

Stabilind timpul de simulare ca fiind de la 0 la 50 (fig. 3.5) se observă că se realizează doar două comparări deoarece au fost generate 51 de numere iar întârzierea de comparare este de 49. Se observă că secvența de informație generată la intrare este refăcută la ieşire (acest lucru se observă cu ajutorul a două afişoare la care am conectat două blocuri de întârziere cu 49 respectiv 50).

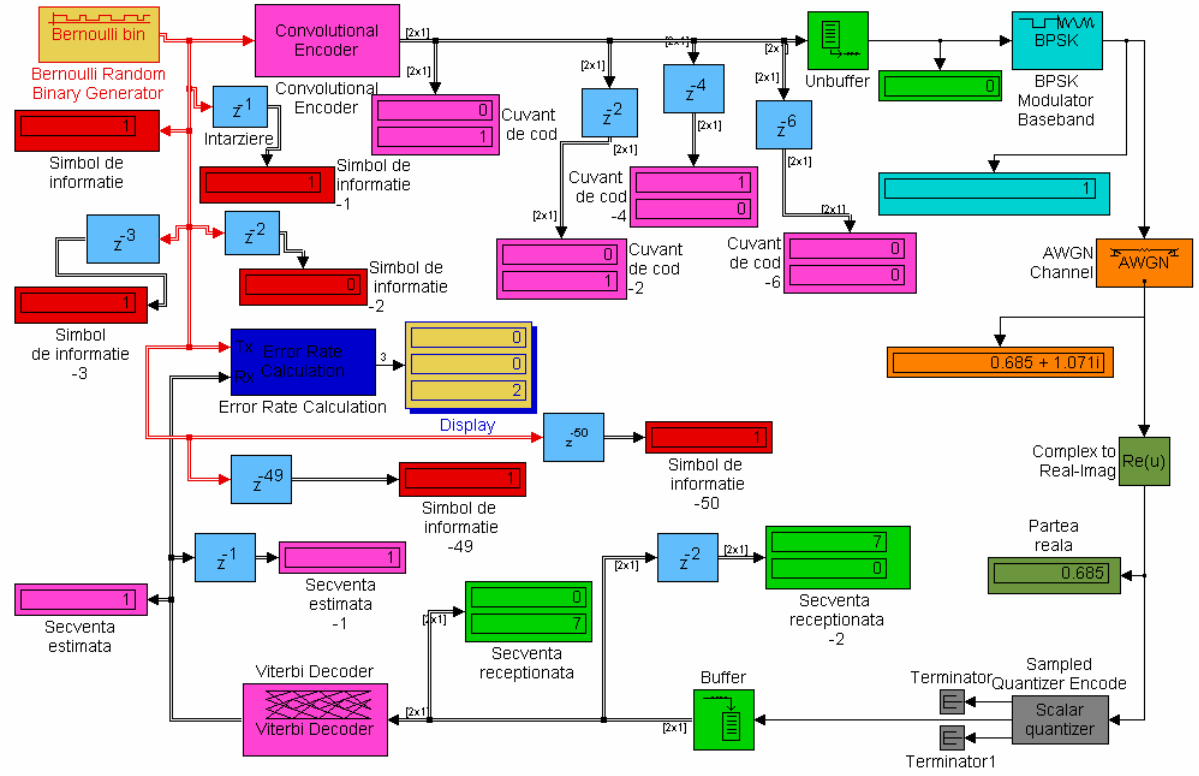


Fig. 3.5

Dacă vom stabili timpul de simulare în intervalul [0,100] vor fi generate 101 numere şi vor fi comparate 52 de numere(fig. 3.6).

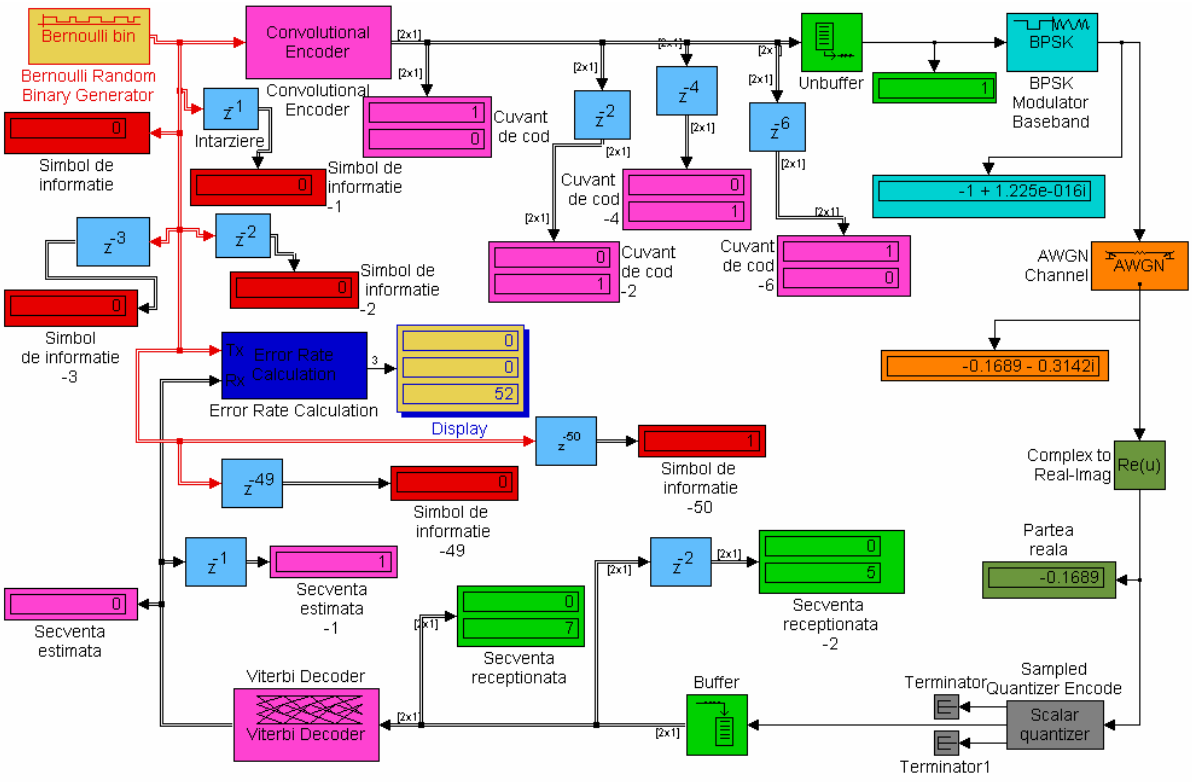

Bernoulli bin
Bernoulli Random Binary Generator
Convolutional Encoder
Convolutional Encoder
Unbuffer
BPSK
BPSK Modulator Baseband
Intarziere
Simbol de informatie
Simbol de informatie -1
Cuvant de cod
Cuvant de cod -4
Cuvant de cod -6
Cuvant de cod -2
-1 + 1.225e-016i
AWGN Channel
AWGN
Simbol de informatie -2
Simbol de informatie -3
Error Rate Calculation
Error Rate Calculation
Display
-0.1689 - 0.3142i
Simbol de informatie -50
Simbol de informatie -49
Complex to Real-Imag
Re(u)
Secventa estimata -1
Secventa estimata
Secventa receptionata
Secventa receptionata -2
Partea reala
-0.1689
Viterbi Decoder
Viterbi Decoder
Buffer
Terminator
Sampled Quantizer Encode
Scalar quantizer
Terminator1

Fig. 3.6