A Viterbi Decoder Architecture for a Standard-Agile and Reprogrammable Transceiver

L. Bissi, P. Placidi*, G. Baruffa and A. Scorzoni

Dipartimento di Ingegneria Elettronica e dell'Informazione (DIEI), Università degli Studi di Perugia

via G. Duranti 93, I-06125 Perugia (Italy)

*Ph.: +390755853636 Fax: +390755853654

pisana.placidi@diei.unipg.it

Abstract

This paper presents a Viterbi Decoder (VD) architecture for a programmable data transmission system, implemented using a Field Programmable Gate Array (FPGA) device. This VD has been conceived as a building block of a Software Defined Radio (SDR) mobile transceiver, reconfigurable on request and capable to provide agility in choosing between different standards. UMTS and GPRS Viterbi decoding is achieved by choosing different coding rates and constraint lengths, and the possibility to switch, at run time, between them guarantees a high degree of programmability. The architecture has been tested and verified with a Xilinx XC2V2000 FPGA, for providing a generalized co-simulation/co-design testbed. The results show that this decoder can sustain an uncoded data rate of about 2 Mbps, with an area occupancy of 46%, due to the efficient resources reuse.

Keywords: Viterbi decoder, software defined radio, Field Programmable Gate Array (FPGA), configurable and programmable architecture.

1. Introduction

In digital communication systems, convolutional encoding coupled with Viterbi algorithm (VA) decoding is widely used for its capability to protect information data from the impairments (e.g. noise, multipath, fading) introduced by the transmission medium. When a potentially corrupted sequence of symbols is received, the VA [1] determines the most likely transmitted sequence by exploiting a maximum likelihood criterion.

The idea behind the Viterbi Decoder (VD) is quite simple, in spite of its inherent implementation difficulty. Moreover, there is a wide gap in complexity with the transmission side, where convolutional encoding can easily be implemented. Since convolutional codes are represented by a state trellis, the decoder is a finite state machine that explores the transitions between states, stores them in a large memory, and comes to a final decision on a

sequence of transitions after some latency due to the constraint length of the input code [1]. Decisions are usually taken by considering the transition metrics among states, which are updated in terms of either Euclidean or Hamming distance with the error-corrupted received sequence.

The performance of convolutional codes strongly depends on their minimum distance, which in turn depends on the constraint length and coding rate. As a consequence, in order to increase the gain with respect to the uncoded case, there is a continuous trend towards increasing such parameters. Thus, complexity may grow up to a limit where classic implementation techniques are no longer viable. Recently, Adaptive Viterbi Decoding (AVD) for the algorithmic part [2], [3] and systolic architectures for the implementation aspects [4], [5] are increasing their popularity in the technical literature. In the AVD approach, only a subset of the states is stored and processed, significantly reducing computation and storage resources at the expense of a small performance loss.

At the same time, a highly complex VD someway loses its advantages, when it is adopted to decode sequences transmitted on a low-noise channel. In this case, low minimum distance codes are more suitable for achieving a good performance, and a higher bit rate can be transmitted by lowering the coding rate. Therefore, programmability can be a viable solution for fast switching between different coding rates when Channel Status Information is available, and a simpler code can be used in place of a stronger code. Adaptive channel coding systems have recently been introduced as means of efficiently allocating power and coding rate [6]. The architecture of the decoder itself, thus, must accommodate for such programmability/reprogrammability, either at the algorithmic or hardware logic level. In general, both requirements can easily be satisfied when an FPGA device is used for this purpose [5], [6], [7], [8].

Moreover, reprogrammability also represents a key aspect of a Software Defined Radio (SDR) terminal [9]. Such a terminal must have an interface with several communication standards, and its decoding capabilities are in large part accomplished by means of software reprogrammable devices [10]. The channel decoding of convolutional codes can be carried out by a programmable architecture VD, which is matched to a particular class of standards and specifications.

In this paper, we present an architecture for a standard-agile, programmable VD that is able to switch between UMTS and GPRS decoding. In our approach, the reconfiguration is not a critical point because we are using the device to implement the decoder only. But in a “real” scenario, where the FPGA houses the whole transceiver digital processing sub-system (such as in Software Radio applications), the possibility to reuse the building blocks for different standards becomes very important in order to minimize efforts during the design process.

Therefore, we propose an architecture that can be shared between two different standards. On the other hand, the programmability of the system can be useful for a particular application, in which the coding rate is modified at run-time during the transmission. The proposed VD can be programmed to deal with convolutional codes of constraint length 5 and 9 and code rate 1/2 and 1/3. In addition, thanks to the reprogrammability of the FPGA, it is possible to implement new features in the VD (e.g. new functionalities or different standards). We begin with presenting the adopted algorithmic architecture of the VD with a constraint length of 9 and a code rate programmable between 1/3 and 1/2, then we describe the modifications needed to manage a constraint length of 5. After that, we show some features of the adopted convolutional codes and the obtained results, in terms of implementation complexity of the decoder. Eventually, we compare the BER performance of our decoder with that of a behavioral model implemented in Matlab.

2. Architecture of the programmable VD

The complexity of the VD exponentially increases with the constraint length K. For increasing values of K, larger hardware resources are required, both in terms of processing power and memory. In this paper, the reuse of resources for reducing the area occupancy and the operating frequency required by the UMTS standard [11] is of primary concern.

Moreover, three-bit soft decision has been adopted as quantized input: this choice represents a good trade-off between complexity and accuracy [12].

The architecture of a Viterbi decoder usually consists of three units, mainly:

1. Branch Metric Unit (BMU), which generates the branch metrics (BMs) and measures the distance between the received symbol and the symbol associated with the transition among trellis states;
2. Add Compare Select Unit (ACSU), which finds the survivor path for each state; the BM of each transition is added to its partial path metric (PM) and the path with the smallest PM is selected as the survivor path. The ACSU also provides a decision bit, which indicates that the survivor path is either the lower or the upper path to the current state;
3. Survivor Memory Unit (SMU), which stores the survivor paths.

The proposed VD is based on the Register-Exchange (RE) method [13], which uses a register for each one of the $2^{(K-1)}$ states. The register records the decoded output sequence associated with the path leading from the initial

to the final state. The RE approach does not require a trace-back and, therefore, it allows for a fast decoding, in comparison with the Trace-Back (TB) approach [2], [4], [5], [14].

It is well known that choosing a survivor path length of 5 times the code constraint length results in a negligible degradation from the ideal decoder performance [12]. Since in our decoder the convolutional code has K=9, a survivor path length equal or larger than 45 is required. Nevertheless, we have chosen a length equal to 52 in order to exploit all the available memory, without wasting resources.

The core of the proposed VD architecture for constraint length equal to 9 is shown in Fig. 1. The block diagram includes the system that provides control signals for the synchronization of the main blocks.

Figure 1

Fig. 1. Block diagram of the core of the Viterbi decoder.

For this architecture, the detailed interconnection among main blocks is depicted in Fig. 2.

Figure 2

Fig. 2. Details of the interconnections among the main blocks of the proposed Viterbi decoder.

A large part of the whole system logic is included in the *Node* block, which is used to evaluate the survivor path metric of the states. The outputs of this block are: the state PM and the decoded output sequence. There are N=16 *Node* blocks (*Node*_0, …, *Node*_15), consecutively numbered and following the order of the states of the trellis, i.e. the generic *Node_k* block (k=0 ,…, 15) analyzes the states labeled with 16*k+i (i=0, ..., 15); each block can manage 16 different states for every input symbol. The *Node* block (Fig. 3) is composed of the following sub-blocks:

- a *ROM*;
- two *BMU*s;
- an *ACSU*;
- a *Control Unit*.

Figure 3

Fig. 3. Block diagram of the *Node* block.

The *ROM* contains the branch output values of the convolutional code. *ROM* addressing is provided by the *Counter* block, which counts the number of times that the *Node* block processes the input symbol. A single bit input (*code_rate*) allows for choosing between 1/3 and 1/2 code rates, and it is the most significant bit of the ROM address. The branch outputs and the input symbols are routed to the *BMU*s that compute the BMs of the upper and lower input branches (*o_u_b*, *o_l_b*), for each state. Since the code rate can be as low as 1/3, the decoder can process three input symbols at the same time. Each *BMU* consists of three sub-blocks, used to calculate the distance between each input symbol and the relevant output branch bits. *BMU*s output is generated by a two-step process in order to increase the computational speed, and a double clock frequency is required.

The *ACSU* inputs are represented by the BMs of the branches entering the state (*BM_u_b*, *BM_l_b*), the PMs, and the decoded output sequences from the previous states (*upper_node*, *lower_node*). This block computes the PM of the two paths entering the state, and the survivor path with lower PM is eventually chosen. The *ACSU* outputs (*node_out bus*) represent the node PM (*metric_out*) and the decoded output sequence (*seq_out* and *data_out*).

The decoded output sequence for each state is the bit sequence of the path, converging to that state, with the lowest PM, which has been left-shifted in order to append the bit associated to the path itself. The *Control Unit* of the *Node* block manages the correct scanning of the trellis, and provides the enable signals (*e_u_BMU*, *e_l_BMU*, *e_ACSU*) to the selected block, for power-saving purposes. Moreover, in order to set up the number of trellis levels, the length of the encoded sequence is required as input.

Fig. 4 shows in detail the connection between *Node* and *RAM* blocks. The outputs of the 16 *Node* blocks (*seq_out*, *metric_out*) are stored in 8 *RAM* blocks.

Figure 4

Fig. 4. Connection between Node blocks and RAM blocks.

The *Node_2j* and *Node_*(2*j*+1) blocks (*j*=0,…,7) outputs are stored in the *RAM_2j_*(2*j*+1) block. The two *RAM* outputs are forwarded to the *Node_j* and *Node_*(*j*+8) blocks, which analyze the states labeled with *j**16+*i* and $j*16+2^{(9-2)}+i$ (*i* = 0, ..., 15). This architecture follows the butterfly structure of the trellis reported in Fig. 5, in order to minimize the number of connections in between. By adopting this strategy, the number of connections among the *Node* and *RAM* blocks is reduced.

Figure 5

Fig. 5. Butterfly structure of the trellis diagram.

Fig. 6 shows an example where the *RAM_*0_1 block stores the *seq_out* and the *metric_out* outputs of the trellis states, and the *RAM_*0_1 outputs are fed back to the *Node_*0 and *Node_*8 blocks. The terms s_x (*x* = 0, ..., 255) in Fig. 6 represent the trellis states whereas t_n and t_{n+1} indicate the trellis levels.

Figure 6

Fig. 6. Equivalence between the trellis diagram and the *Node* and *RAM* blocks connection.

Each *RAM* block has been implemented using two dual-port RAM blocks, configured with a 32 bits data bus in order to use 52 and 12 bits, respectively, for storing the *seq_out* and *metric_out* of each state. Since the *RAM* blocks feature concurrent read/write capabilities, they work at twice the clock frequency. Each *Node* block stores its outputs in 16 contiguous *RAM* locations. Moreover, the memory is divided into two sections, where the outputs of the two previous level states are stored. In fact, the input of some states must be the PM of the previous level state, after the current level PM has been stored.

After 52 input symbols, the *Min_Detector* block is enabled, and it selects the path with minimum PM among all the state paths, thus providing a bit (*valid_out*) to point out if the decoded output (*data_out*) is available.

The *Min_Detector* computes the minimum PM between the state paths associated to a *Node* block.

The Digital Clock Manager (*DCM*) block drives the clocks and provides three different signals: *clk* with a frequency $f=f_0$, *clk2* with $f=2f_0$ and *clk16* with $f=f_0/16$. The last signal has been used to synchronize the decoded bits.

In order to tailor the VD architecture to include both UMTS and GPRS decoding schemes, a *UMTS_GPRS Control Unit* is required, as shown in Fig.7, and with a single-bit input (*umts_gprs*) it is possible to choose between the UMTS and GPRS decoding. For convolutional decoding with constraint length K=5, only the *Node*_0 block has been used to run the path metric of the states, the other *Node* blocks being disabled, and only one *ROM* stores the branch outputs. The *UMTS_GPRS Control Unit* block is also used to manage the scanning of the trellis for the selected standard. Power consumption has been minimized by enabling the blocks involved in the decoding operation only. The outputs of the *Node* block are stored in a *RAM* block. A simplified *Min_Detector* block selects the path with minimum PM among all nodes.

Figure 7

Fig. 7. Block diagram of the UMTS and GPRS Viterbi decoder.

The change in the VD constraint length is performed by a single-bit input and an initialization procedure is required, as reported in Fig. 8. Thus, a fast and efficient method for the programmability of the decoder is achieved. As stated before, this is useful in all those applications where multi-standard terminals are required, and agility in switching between them is of fundamental importance. Furthermore, if channel conditions change, the coding/decoding process could adapt its complexity and protection capabilities accordingly.

Figure 8

Fig. 8. Flow chart of the VD operations.

In addition, due to the flexibility of the proposed architecture, novel and emerging telecommunication standards adopting Viterbi decoding at the receiver side (with similar constraint lengths) can be considered and implemented with minor modifications.

3. Implementation

3.1 Specifications

The proposed architecture has been verified with reference to the convolutional codes adopted by the UMTS and GPRS standards. The UMTS cellular standard uses convolutional codes with constraint length K=9 [11]. A data sequence of L bits ($L \leq 504$) is encoded, and 8 null tail bits are added at the end of the uncoded data block, in order to reset the convolutional encoder to a starting state. This code can be represented using a trellis of 256 states, and a maximum depth of 512 levels. Each state has two incoming and two outgoing branches. In the UMTS standard, two different code rates are used, i.e. rate 1/3 with generating polynomials $G_0=557_{OCT}$, $G_1=663_{OCT}$, $G_2=771_{OCT}$, and rate 1/2 with generating polynomials $G_0=561_{OCT}$, $G_1=753_{OCT}$.

In this paper, we do not consider the turbo-coding option.

The GPRS standard, instead, uses four different coding schemes (CS) [15]:

- CS-1: 224 information bits and 4 tail bits;
- CS-2: 290 information bits and 4 tail bits;
- CS-3: 334 information bits and 4 tail bits;
- CS-4: 456 information bits.

CS-1, CS-2 and CS-3 use a convolutional code with constraint length K=5 and rate 1/2, with generating polynomials $G_0=23_{OCT}$, $G_1=33_{OCT}$. This code is represented by a trellis with 16 states and, at most, a depth of 338 levels.

3.2 Software and Hardware partitioning

In order to take advantage of both software and hardware implementations, the VD has been designed with a cooperation of software and hardware modules in mind.

The VD has been designed using VHDL, implemented and tested on an FPGA device. To this purpose, a Matlab reference model of the channel coding communication system has been used (Fig. 9).

Figure 9

Fig. 9. Hardware and software partitioning.

This testbed is used to assess the channel decoding sub-system performance only, so there is no need to simulate the modulations used in the relevant standards. To this aim, we have chosen a simple Binary Phase Shift Keying (BPSK) baseband mapping scheme, and evaluated hard- or soft-decision decoding over an Additive White Gaussian Noise (AWGN) channel. A hard-decision VD has been simulated using a Matlab model.

The random-bit generator creates a bit sequence of variable length. The convolutional encoder can be operated with the generating polynomials of either the UMTS or the GPRS standard. The symbol mapper converts a coded bit into its BPSK baseband equivalent. The output of the mapper is transmitted over the AWGN channel, and the received noisy symbols are converted to three soft-bits before being decoded.

A Matlab script compares the output of the hardware VD with the original sequence, thus evaluating the Bit Error Rate (BER) for different values of E_b/N_0, where E_b is the Energy per Bit and N_0 is the Noise Power Spectral Density. In order to verify the correct implementation of the proposed VD, a three bit soft-decision VD Matlab model has been implemented.

The VD architecture has been described using the VHDL language, and targeted to a Xilinx FPGA (Virtex II XC2V2000) device, housing large amounts of 18 kbit block SelectRAM [16] (configurable in several bit- and address-widths). In order to minimize the use of configurable logic blocks (CLBs), dual port RAMs have been selected. Three different clock rates, managed by a Digital Clock Manager (DCM) [16], are required in order for the system to operate.

4. Results

The experimental set-up and the partitioning between software and hardware are shown in Fig. 10. The VD has been tested with a Nallatech XtremeDSP Development Kit [17], and the relevant results have been compared with both functional and Matlab simulations.

Figure 10

Fig. 10. Experimental set-up.

Data exchange between hardware (FPGA) and software (CPU) processing units is performed either through a PCI or USB bus. This solution is fundamental not only when a faster co-simulation method is needed to validate architectures and algorithms, but also when reconfigurability of the communication terminal is of concern.

The maximum clock frequency (*clk*) achieved for UMTS and UMTS/GPRS after placement and routing (Fig. 11) is equal to 32.26 MHz. Thus, the decoder is able to supply raw data at a maximum rate of 2.016 MHz (corresponding to a maximum decoded bit rate of 2.016 Mbps), which is compliant with both the UMTS and GPRS standards rates. It should be remarked that the operating frequency and the bit rate are different, because each block has to manage sixteen states of the trellis (as already explained in Section 2). The proposed Viterbi decoder is able to run faster than the VDs implemented using the trace-back method reported in refs. [2], [4], [5]. In Table I, the required resources for the two different decoder structures are summarized.

TABLE I

Resources used in the FPGA device (percentile figures are referred to the whole device area).

Performance parameters (Virtex II XC2V2000)	**UMTS only VD**	**UMTS/GPRS VD**
Logic Utilization:		
Number Slice Registers	3490 (16%)	3501 (16%)
Number of 4 input LUTs	7269 (33%)	8170 (38%)
Logic Distribution:		
Number of SLICEs	4732 (44%)	4966 (46%)
Number of 4 input LUTs	8014 (37%)	8881 (41%)
Number of bonded IOBs	38 (8%)	39 (8%)
Number of DCMs	1 (12%)	1 (12%)
Number of Block RAMs	33 (59%)	33 (59%)
Total equivalent gate count	2,253,595	2,256,522
Max. clock frequency	32.256 MHz	32.256 MHz
Max. decoded bit rate	2.016 Mbps	2.016 Mbps

We point out that the resources overhead due to the reconfigurable implementation (UMTS/GPRS VD) in comparison with the fixed implementation (UMTS only VD) is just 2%. This result is fully in agreement with the approach described in ref. [7]. As concerns area occupancy, we used a generic architecture, therefore we can appreciate a significant difference only if we consider an architecture (e.g. as reported in ref. [18]) that is optimized for maximum speed and with K=7 instead of K=9. A summary of the performance of the referenced VD architectures has been reported in Table II. These figures are compared with the results obtained in this paper.

Figure 11

Fig. 11. Results of placement (a) and routing (b) operations.

TABLE II

Summary of the performance of VD architecures (Label "—" means the information is not available).

	Guo [2], [4]	**Tessier [5]**	**Chadha [7]**	**Han [18]**	**Proposed architecture**
Area occupancy	2793/5120 slices (54%)	1469/12288 slices (12%)	89407/888439 gates (10%)	8090/10752 slices (75%)	4966/10752 slices (46%)
Throughput	78 kbps	240.3 kbps	19.7 Mbps	—	2.016 Mbps
Programmability / overhead	No	No	Yes / 2.9%	No	Yes / 2%
Constraint length (K) and code rate (R)	K=9 R=1/2	K=9	K=3-7 R=1/2-1/3	K=7 R=1/2	K=5-9 R=1/2-1/3
Operating clock	40 MHz	7.6 MHz	—	13.3 MHz	32.3 MHz
Coding gain at a BER of 10^{-5}	5 dB	3.1-3.7 dB	—	—	3.5 dB

Fig. 12 plots the Bit Error Rate (BER) vs. E_b/N_0, obtained decoding data sequences of 504 bits, encoded with the code rate R=1/3 UMTS convolutional code. The different symbols and curves are relevant to a three-bit soft decision VD implemented in hardware (3-bit Soft decision VD XtremeDSP Development Kit), the Matlab model (3-bit Soft decision VD Matlab model), and the VHDL behavioral simulation (3-bit Soft decision VD VHDL simulations). In addition, a behavioral hard decision implementation (Hard decision VD) has been reported, too. These results compare well with those of ref. [19].

Figure 12

Fig. 12. Bit error rate (BER) versus E_b/N_0 for the UMTS ($G_0=557_{OCT}$, $G_1=663_{OCT}$, $G_2=771_{OCT}$) standard.

Fig. 13 shows the BER vs. E_b/N_0, obtained when decoding data sequences of 334 bits encoded by the GPRS convolutional code. The obtained results are comparable with those of ref. [12].

Figure 13

Fig. 13. Bit error rate (BER) versus E_b/N_0 for the GPRS ($G_0=23_{OCT}$, $G_1=33_{OCT}$) standard.

The good agreement among the curves validates the overall performance of the adopted architecture.

5. Conclusion

In this paper we have presented a novel, low hardware demanding architecture for a standard-agile, programmable Viterbi Decoder. This decoder can easily switch between UMTS and GPRS decoding, and vice-versa, making it suitable for Software Radio applications. The introduction of GPRS decoding capability entails only a small increase in terms of area with respect to the UMTS decoder alone. The efficient resource reuse allowed us to obtain a relatively small area occupancy. The performance of the implemented decoder is in good agreement with that obtained by means of functional and Matlab simulations, and with similar architectures reported in literature.

Moreover, the obtained decoded bit rate of 2 Mbps is compliant with both the UMTS and GPRS standard rates.

References

[1] G. D. Forney, "The Viterbi Algorithm", Proceedings of the IEEE, vol. 61, no. 3, March 1973, pp. 268-278.

[2] M. Guo, M. Omair Ahmad, M. Swamy, and C. Wang, "A low-power systolic array-based adaptive Viterbi decoder and its FPGA implementation", in Proc. of ISCAS '03, vol. 2, 25-28 May 2003, pp. 276-279.

[3] X. Qin, M. Zhu, Z. Wei, and D. Chao, "An adaptive Viterbi decoder based on FPGA dynamic reconfiguration technology", in Proc. of ICFPT'04, 2004, pp.315-318.

[4] M. Guo, M. Ahmad, M. Swamy, and C. Wang, "FPGA Design and Implementation of a Low-Power Systolic Array-Based Adaptive Viterbi Decoder", in IEEE Trans. on Circuits and Systems I, Vol. 52, February 2005, pp. 350-365.

[5] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder", IEEE Trans. on VLSI Systems, Vol. 13, April 2005, pp. 484-488.

[6] Eun-A Choi, Ji-Won Jung and Nae-Soo Kim, "FPGA realization of adaptive coding rate trellis-coded 8PSK system", Proceedings of PIMRC 2003, vol. 1, 7-10 Sept. 2003, pp. 702 – 706.

[7] K. Chadha, and J. Cavallaro, "A reconfigurable Viterbi decoder architecture", in Proc. of 35th Asilomar Conference on Signals, Systems and Computers, vol. 1, 4-7 Nov. 2001, pp. 66-71.

[8] Z. Cheng, S. Khawam, and T. Arslan, "Domain specific reconfigurable fabric targeting Viterbi algorithm", in Proc. of ICFPT'04, 2004, pp. 363-366.

[9] J. Mitola, "Technical challenges in the globalization of software radio", IEEE Personal Commun., Vol. 6, August 1999, pp 84-89.

[10] M. Cummings, and S. Haruyama, "FPGA in the software radio", IEEE Commun. Magazine, Vol. 37, February 1999, pp. 108-112.

[11] 3GPP TS 25.212, “Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)”, http://www.3gpp.org.

[12] J.G. Proakis, Digital Communications, 3rd Edition, Mc Graw Hill, 1995, pp. 506-511.

[13] D. A. F. El-Dib, and M. I. Elmasry, “Low-power register-exchange Viterbi decoder for high-speed wireless communications”, Proc. 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002, May 2002), pp. 737-740.

[14] T. K. Truong, M.-T. Shih, I. S. Reed, and E. H. Satorius, “A VLSI design for a trace-back Viterbi Decoder”, IEEE Transactions on Communications, 40, no. 3, March 1992, pp. 616-624.

[15] 3GPP TS 03.64 V8.12.0 (2004-04) "3rd Generation Partnership Project; Technical Specification Group GSM/EDGE Radio Access Network; General Packet Radio Service (GPRS); Overall description of the GPRS radio interface; Stage 2 (Release 1999)", http://www.3gpp.org/ftp/Specs/html-info/0364.htm

[16] Xilinx, “Virtex™-II Platform FPGAs: Detailed Description”, DS031-2 (v3.1) Product Specification, October 2003, http://www.xilinx.com

[17] “XtremeDSP Development Kit User Guide NT107-0132, Issue 9”, http://www.nallatech.com.

[18] Jae-Sun Han; Tae-Jin Kim; Chanho Lee, “High performance Viterbi decoder using modified register exchange methods”, Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on, Volume 3, 23-26 May 2004 Page(s):III - 553-6 Vol.3

[19] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R.-H. Yan, “A Unified Turbo/Viterbi Channel Decoder for 3GPP Mobile Wireless in 0.18-µm CMOS”, IEEE Journal of Solid-State Circuits, vol. 37, no. 11, pp. 1555–1564, November 2002.

Figure1

Click here to download high resolution image

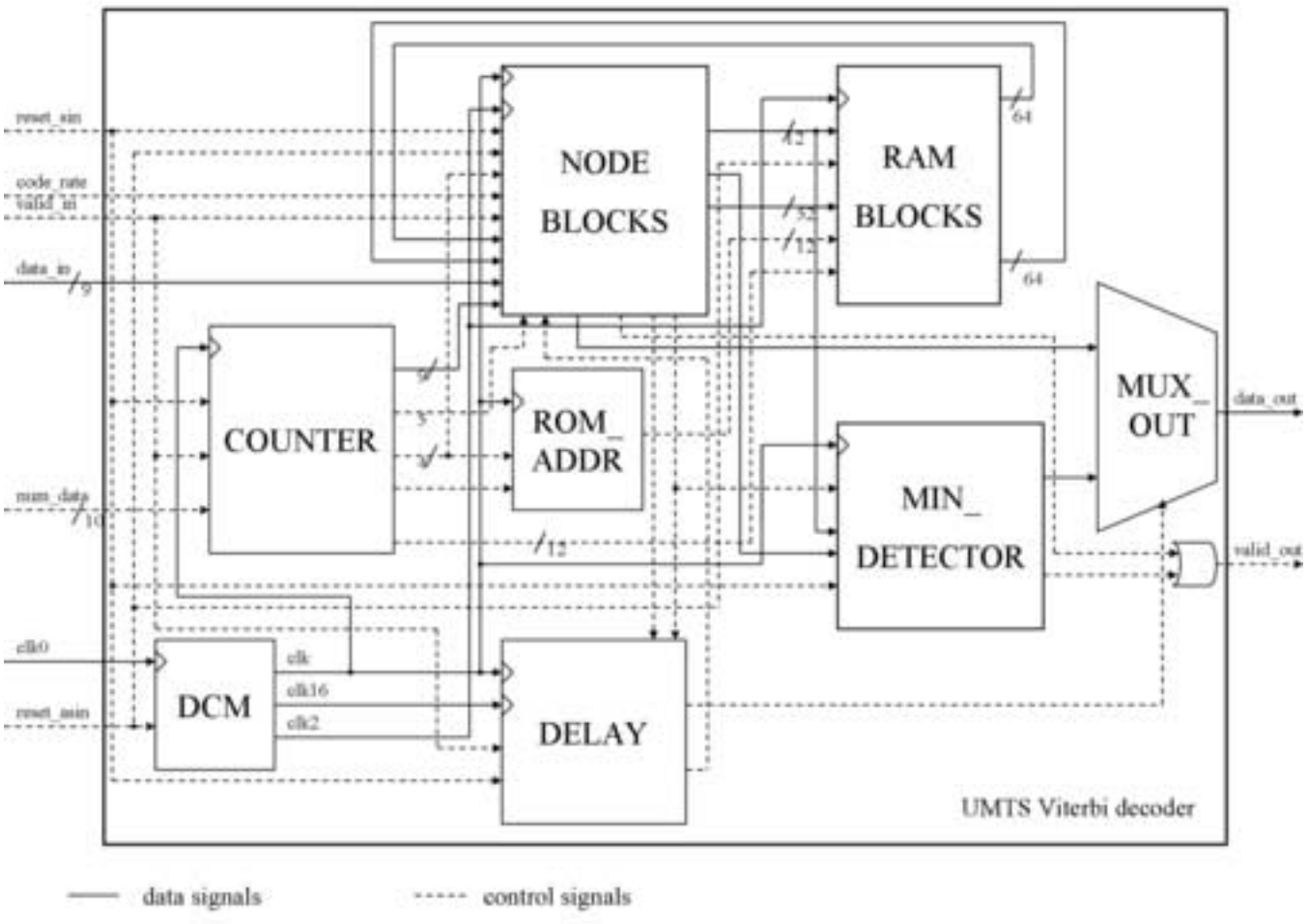

Figure2

Click here to download high resolution image

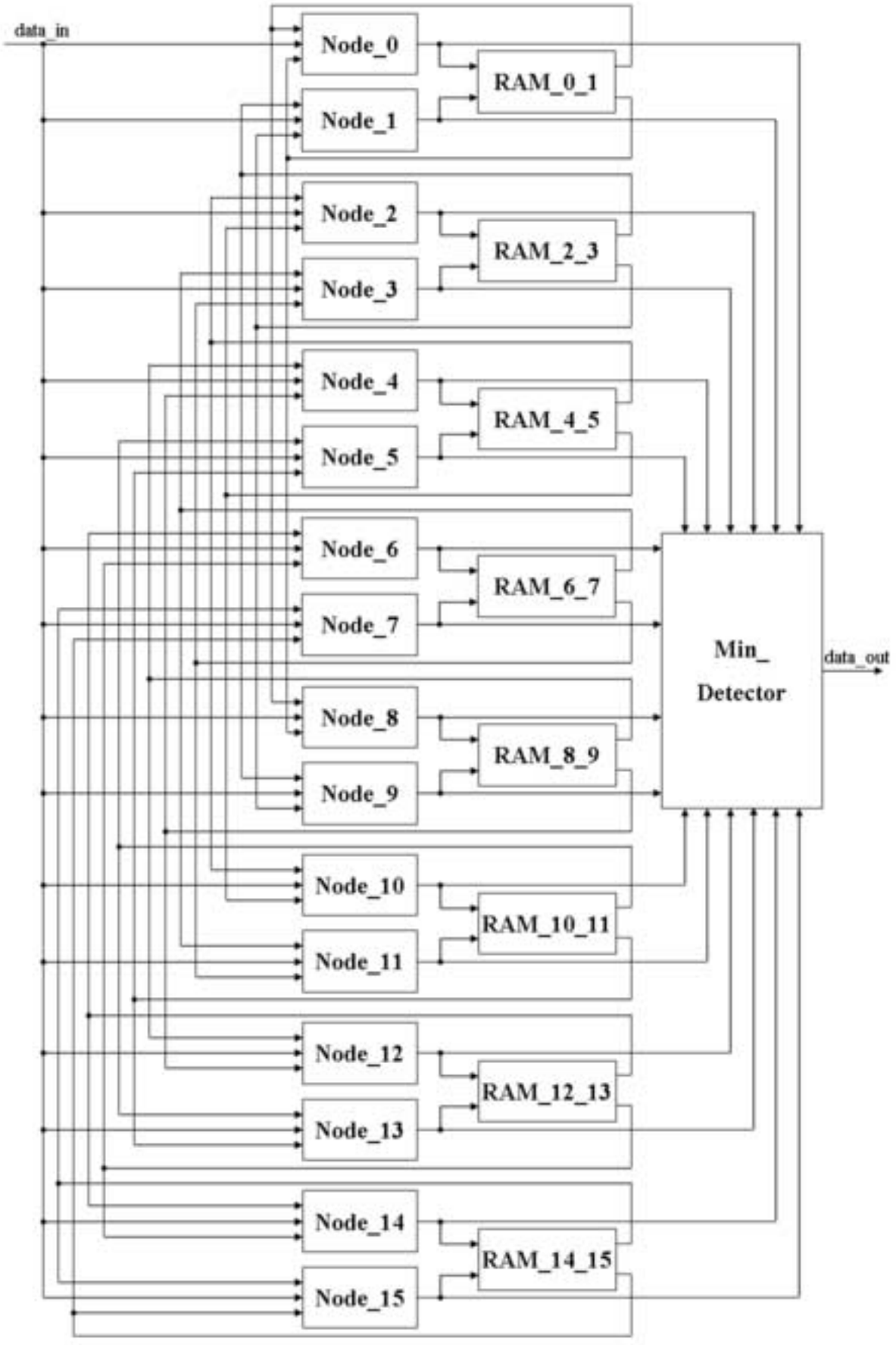

Figure3

Click here to download high resolution image

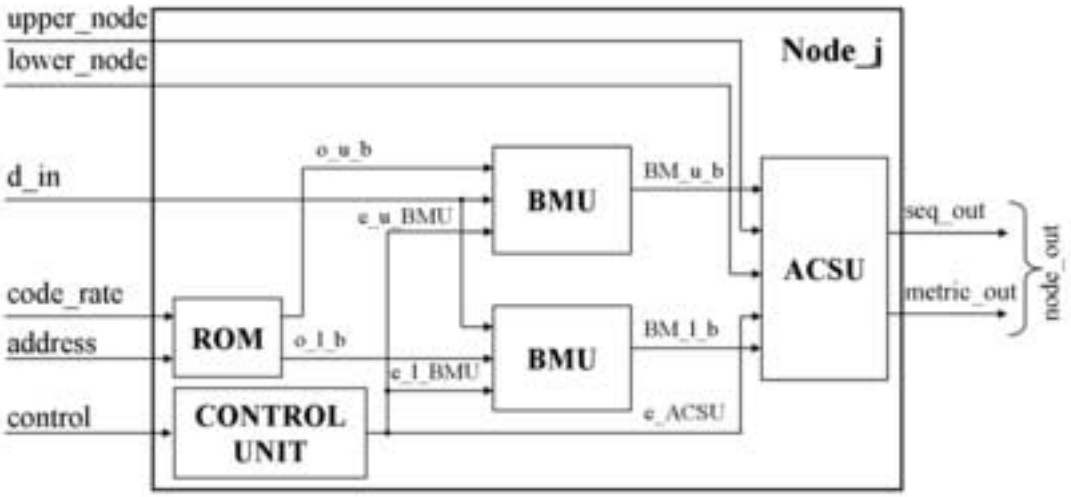

Figure4

Click here to download high resolution image

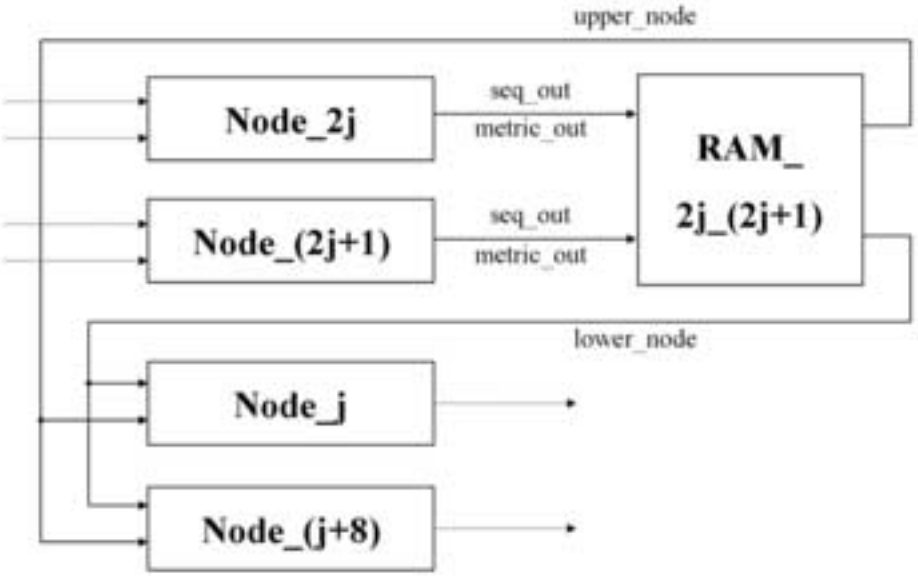

Figure5

Click here to download high resolution image

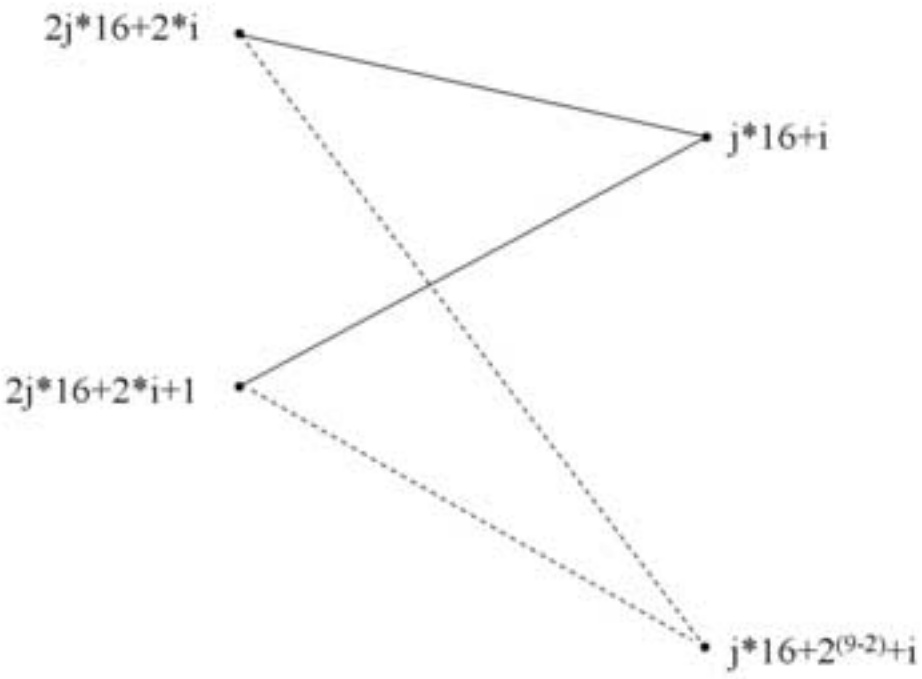

Figure6

Click here to download high resolution image

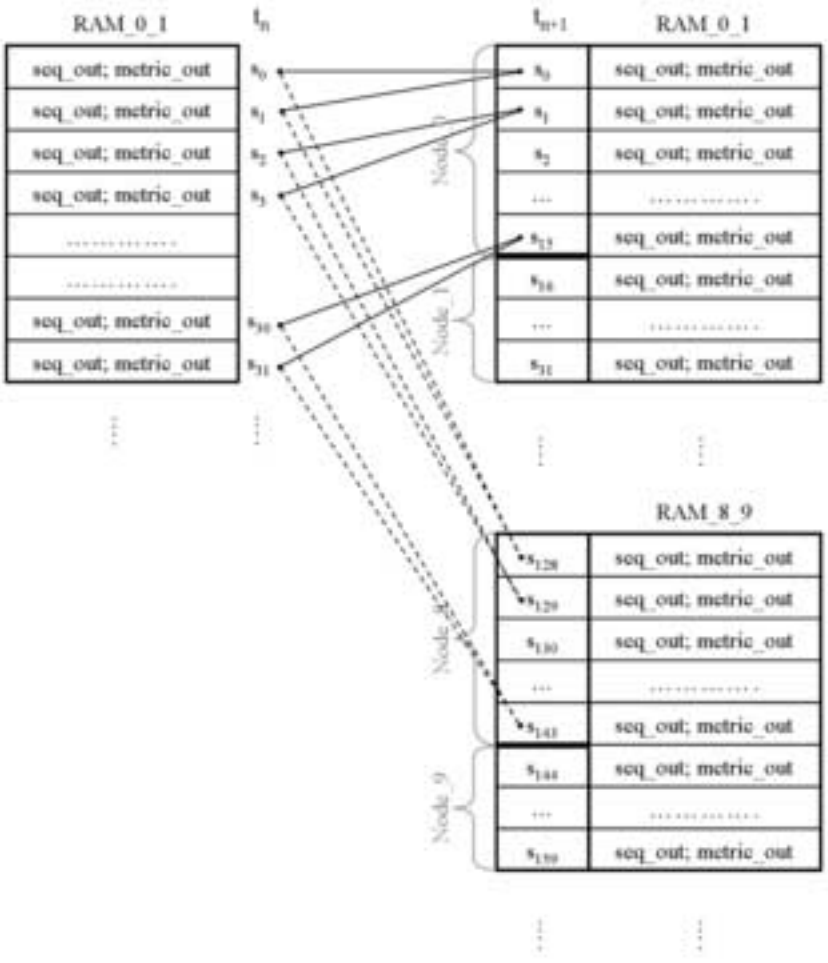

Figure7

Click here to download high resolution image

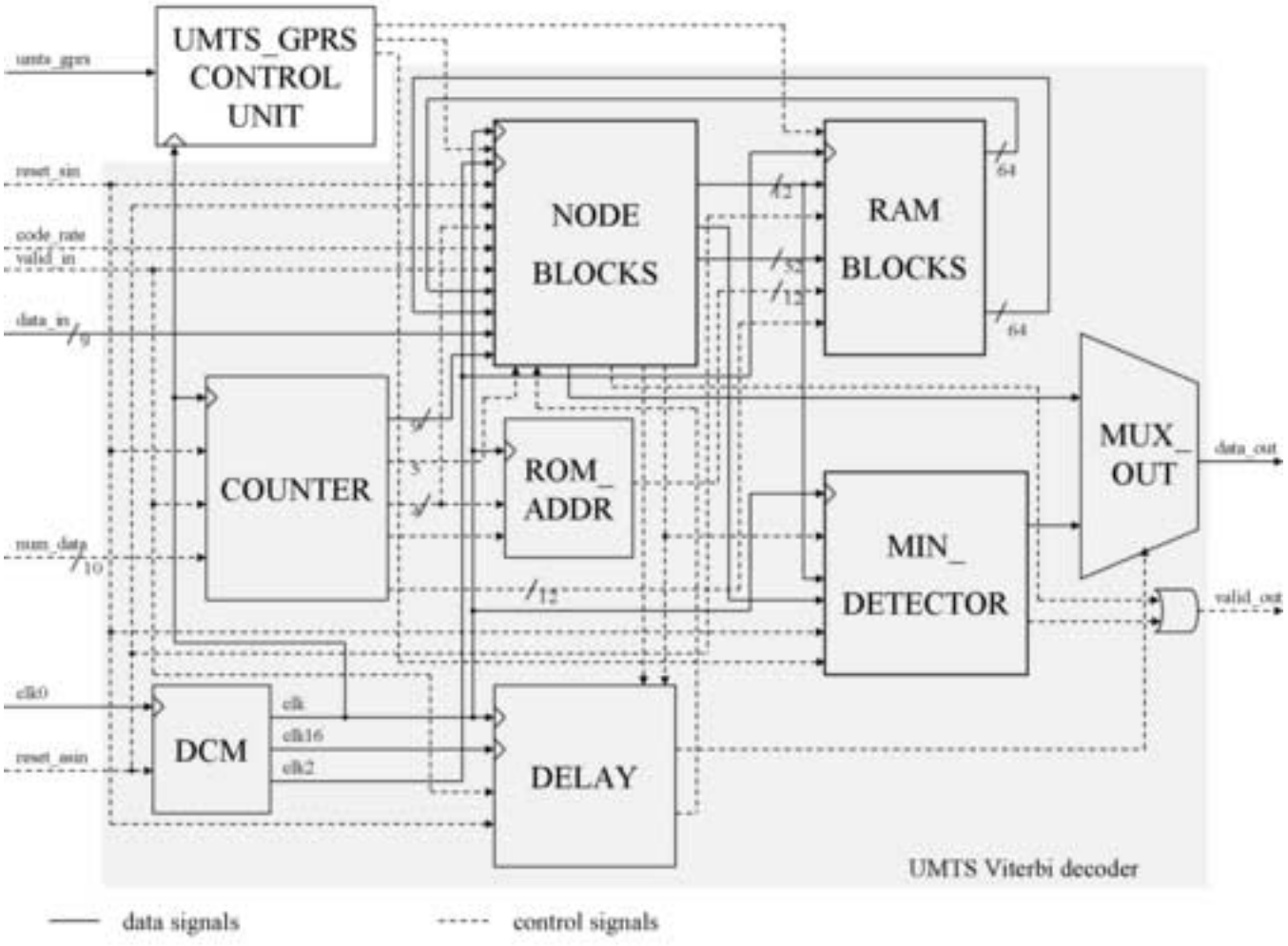

Figure8

Click here to download high resolution image

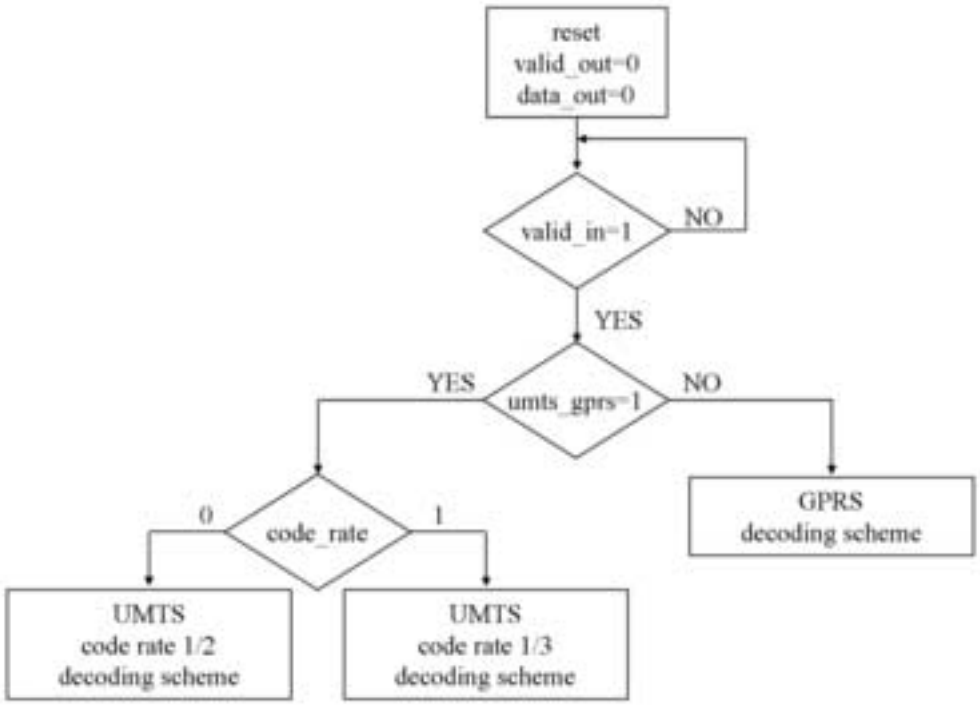

Figure9

Click here to download high resolution image

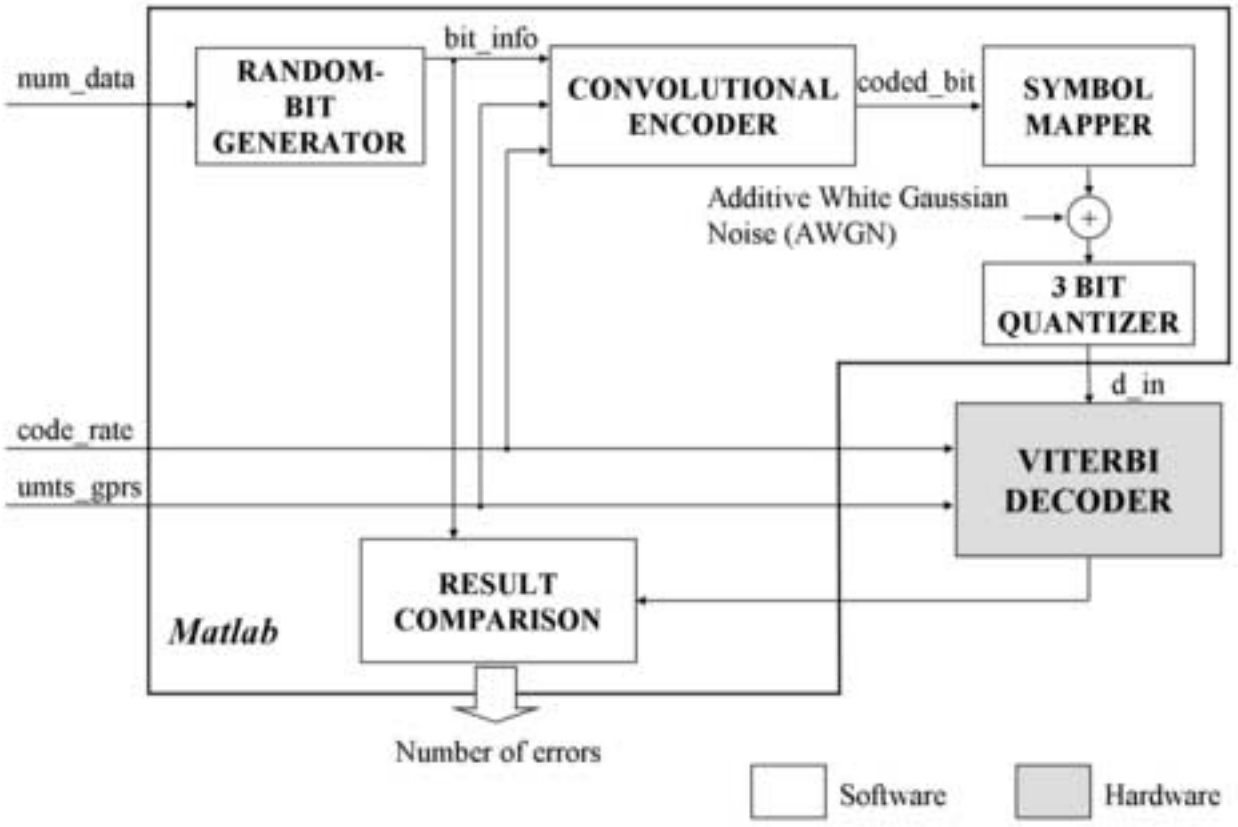

Figure10

Click here to download high resolution image

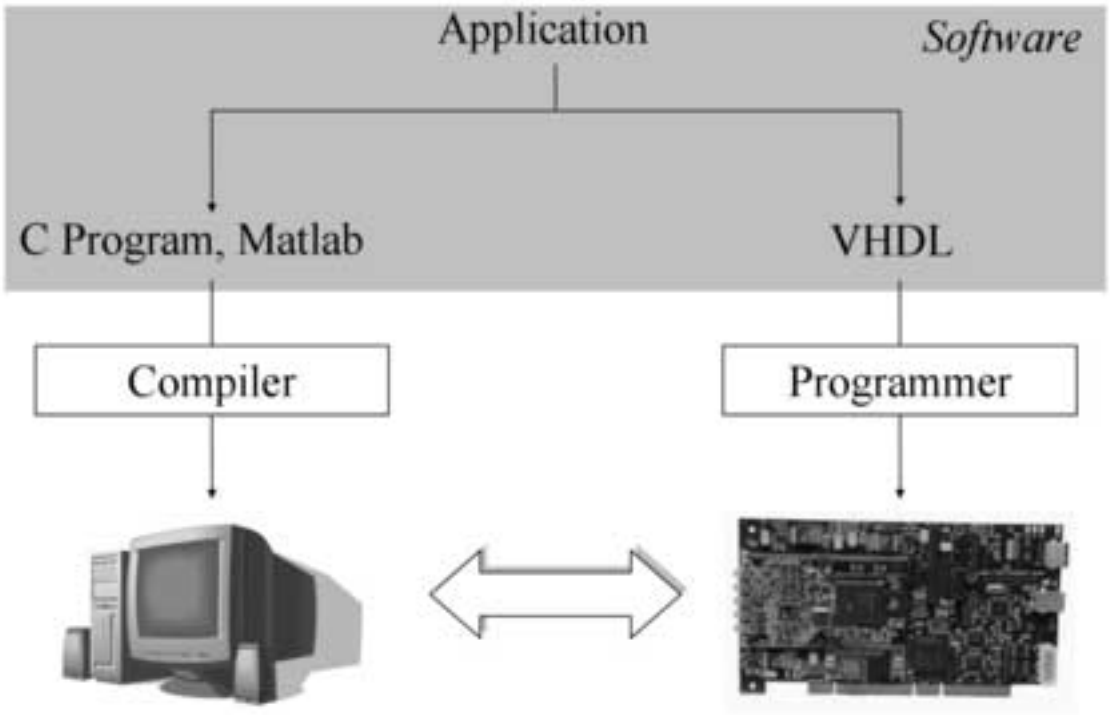

Figure11a

Click here to download high resolution image

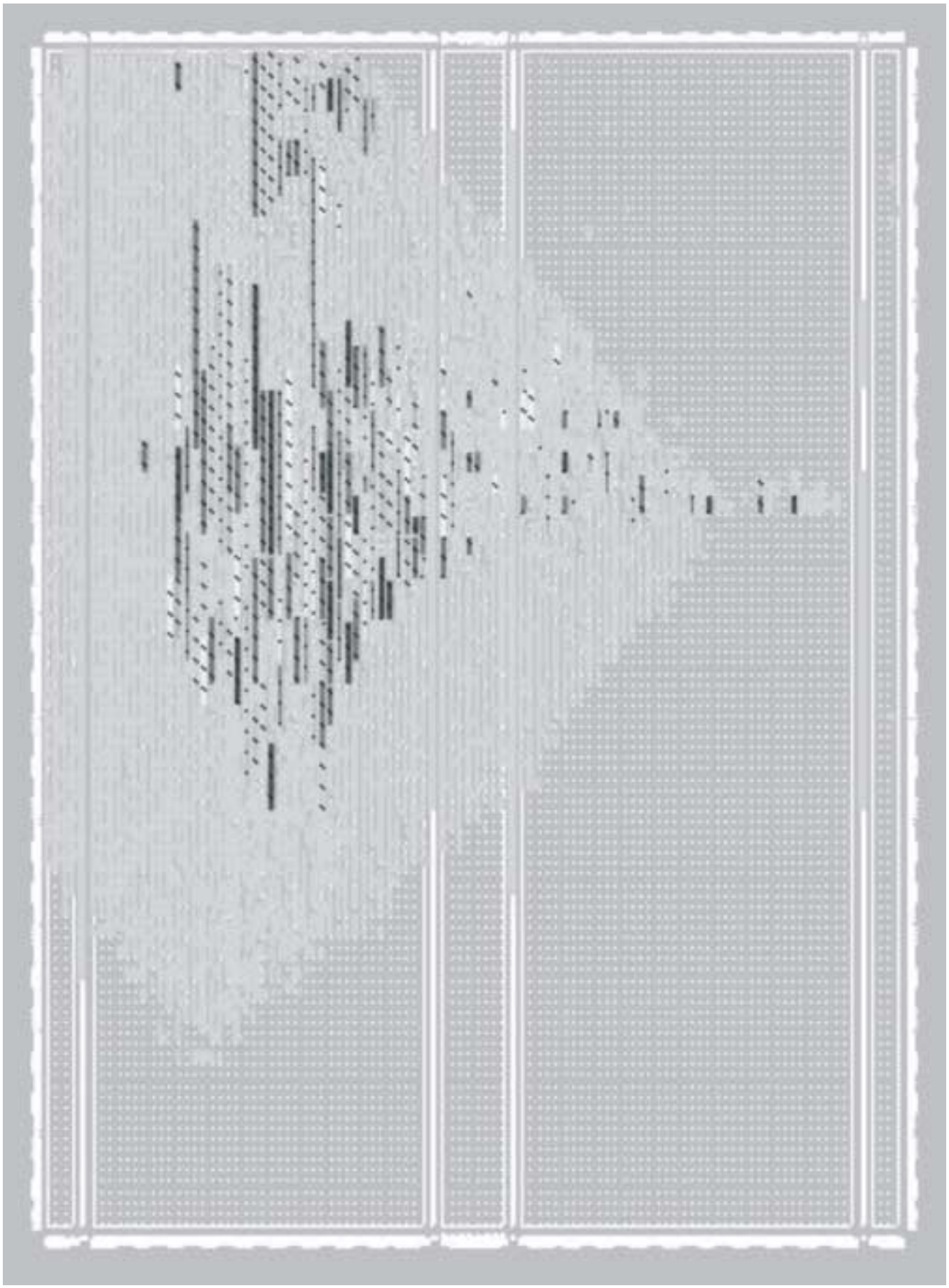

Figure11b

Click here to download high resolution image

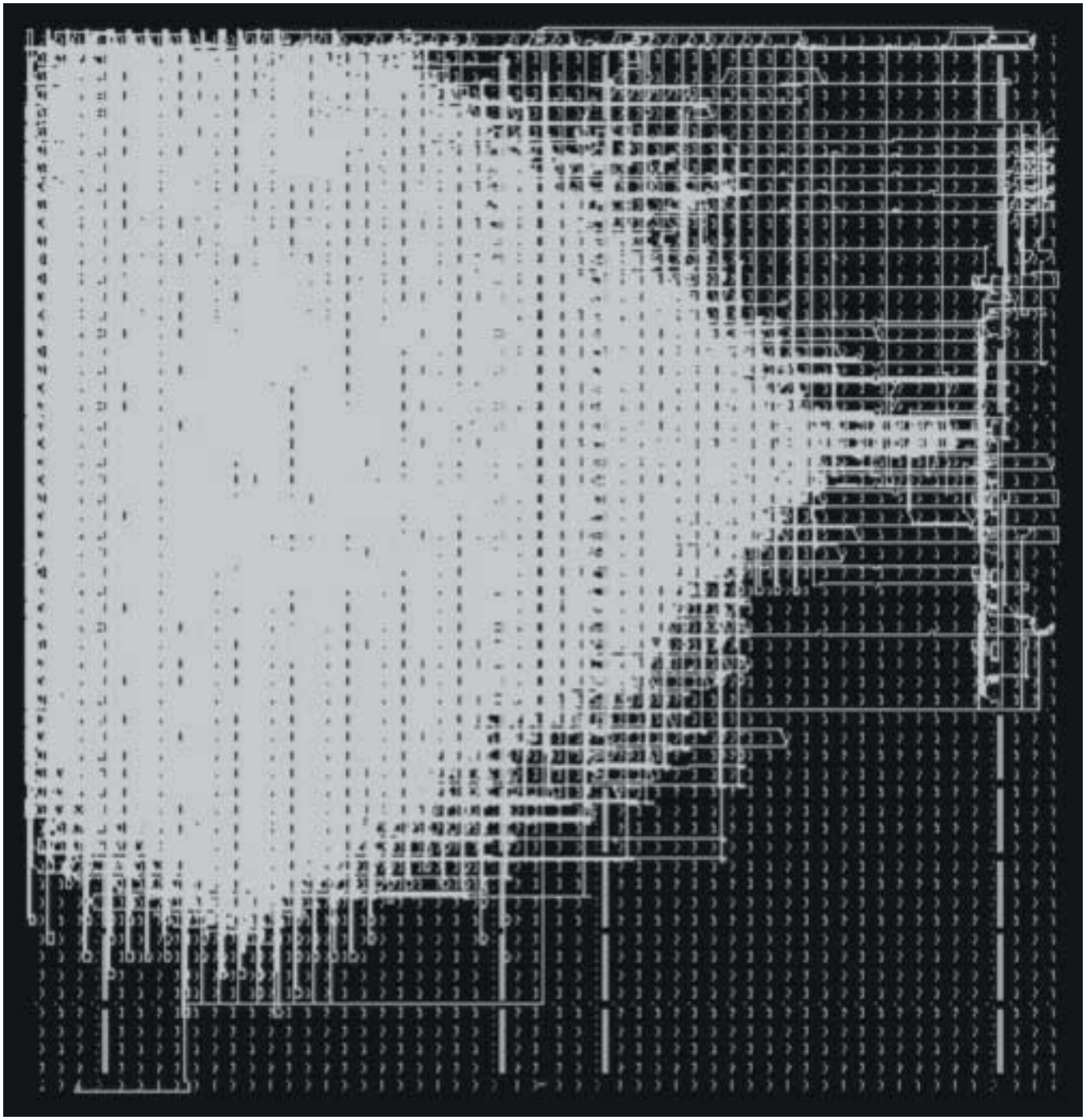

Figure12

Click here to download high resolution image

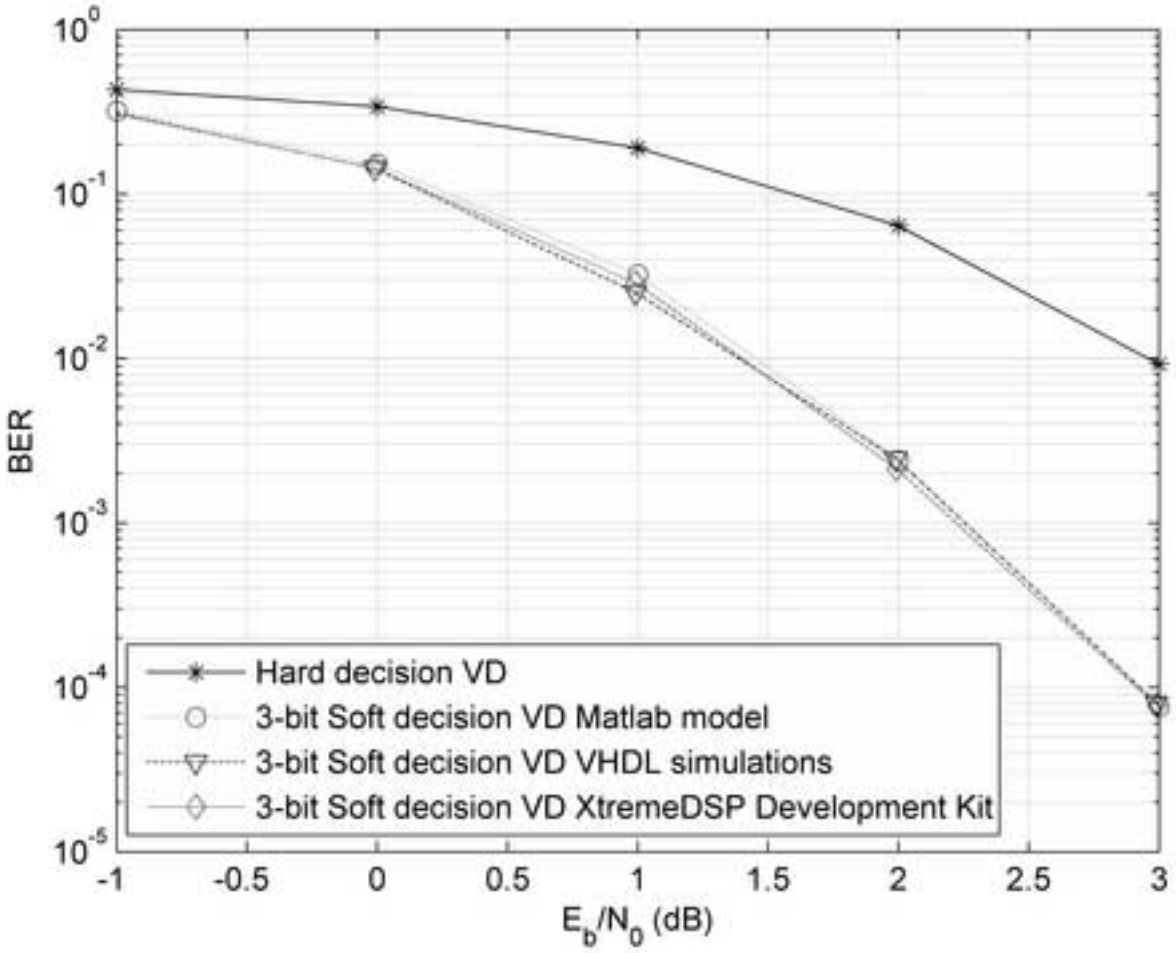

Figure13

Click here to download high resolution image