IMPLEMENTATION OF MESSAGE-PASSING ALGORITHMS FOR THE ACQUISITION OF SPREADING CODES

Massimo Rovini, Fabio Principe, Luca Fanucci, and Marco Luise

Department of Information Engineering - University of Pisa
Via G. Caruso 16, I-56122 Pisa - Italy
{*massimo.rovini, fabio.principe, luca.fanucci, marco.luise*}@iet.unipi.it

ABSTRACT

A new technique to acquire *pseudo-noise* (PN) sequences has been recently proposed in [1] and [2]. It is based on the paradigm of *iterative Message Passing* (iMP) to be run on loopy graph. This technique approximates the *maximum-likelihood* (ML) estimator, providing a sub-optimal algorithm that searches all possible *code phases* in parallel, at low complexity and fast acquisition time. This work is addressed to the design of the architecture of an iMP detector, following the implementation methodologies typical of standard *low-density parity-check* (LDPC) decoders, and demonstrates its benefits in terms of acquisition time and complexity, compared with standard techniques.

Index Terms— Message passing, low-density parity-check codes, code division multiple access, decoder architecture, system design

1. INTRODUCTION

THE ACQUISITION of spreading codes is an essential stage of a DS/SS receiver. All standard detection algorithms are based on the correlation between the received PN code and its local replicas that are shifted until the right alignment is achieved, [3]. Thus, they are inadequate to acquire long spreading codes, because they can either yield fast detection at high complexity (e.g., full parallel search), or have very-low complexity but extremely slow (e.g., serial search).

In this context, new methods based on modern coding theory have recently been presented in [1] and [2], to acquire PN sequences. They are based on running *iterative Message-Passing* (iMP) algorithms on graphical models (e.g. *Tanner/Wiberg Graphs*) with cycles, [4], so approximating the ML method. These sub-optimal estimators search all possible code phases in parallel with a complexity typically lower than the full parallel implementation, and an acquisition time shorter than the serial algorithm.

Considering the studies and results reported in [1] and [5], this paper is addressed to design a hardware implementation of an iMP detector. Specifically, our approach is based on the design of an iMP decoder, which implements a very-effective activation schedule, named *layered* decoding. Then, a vectorized architecture is described, and its performance and hardware complexity are thoroughly analyzed. Furthermore, the analysis is enriched by comparing this novel implementation with classical architectures (parallel and serial searches) in terms of correct/missed/wrong detection probabilities and complexity.

2. SYSTEM MODEL

A basic base-band (BB) model of a DS/SS communication system during the acquisition stage is sketched in Fig. 1. It is made up of: a *BB transmitter*, a *communication channel*, and a *BB receiver*.

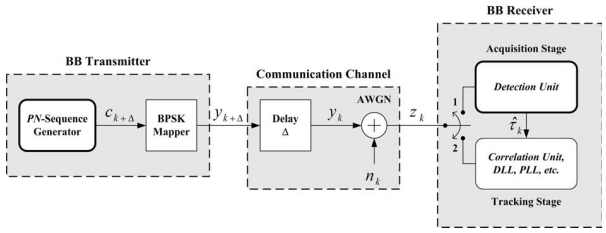


Fig. 1. Simplified DS/SS communication system model.

The BB transmitter is basically a PN sequence generator, which yields a binary sequence $\mathbf{c}$ (with elements $c_k \in \{0,1\}$), and a BPSK mapper that outputs the correspondent antipodal sequence $\mathbf{y}$ (with symbols $y_k = (-1)^{c_k}$). Only *linear feedback shift register* (LFSR) generators are taken into account in this paper.

A common way to characterize an r-stage LFSR is providing its *generating polynomial* as $P(D) = g_0 + g_1\, D + \ldots + g_r\, D^r$, where D is the unit delay operator, g_i is the i^{th} *tap*, and $r = \deg[P(D)]$ the polynomial degree. Then, at the generic time k, the parity equation $\bigoplus_{i=0}^{r} g_{r-i} \cdot c_{k+i} = 0$ is satisfied by any subset $\{c_k, \ldots, c_{k+r}\}$ of the LFSR sequence $\mathbf{c}$, with $\oplus$ denoting modulo-2 sum.

M-Sequences are a subset of the LFSR family, because their generating polynomials are primitive, [3]. This implies that their period N, is the maximum achievable with an r-stage LFSR generator ($N = 2^r - 1$), and, furthermore, that every m-sequence is univocally identified by its primitive polynomial. Indeed, the initial word of the SR (except for the *forbidden* all-zero word) only produces a cyclical shift of the code.

The incoming BB spreading signal is modeled as a *coherent pilot* signal (without data modulation and carrier frequency/phase offset) $z_k = \sqrt{E_c} \cdot y_k + n_k = \sqrt{E_c} \cdot (-1)^{c_k} + n_k$ where z_k is a noisy sample at the receiver side and n_k is an *additive white gaussian noise* (AWGN) with mean value 0 and variance $N_0/2$. This is a simplified representation, which is widely used in SS literature to *isolate* the issue of spreading-sequence acquisition (see [2] and [3]).

All traditional detection techniques used in DS/SS receivers are correlation-based algorithms. Specifically, they shift local replicas of the transmitted PN code, and correlate them with the received signal until the right alignment is got. When this happens, a suitable decision unit should detect the correlation peak, taking the correspondent code shift as an estimate of the incoming code phase. A *verification stage* is sometimes run to check the correct detection on a longer estimation time. If the last check is verified, the tracking stage is launched, otherwise the local code is cyclically shifted and a new acquisition try is carried out.

The most common designs of these algorithms are full parallel, simple serial, and hybrid implementations [3].

In this context a new method to acquire spreading sequences, based on the iterative decoding theory of channel codes, has been proposed in [1] and [2]: indeed, it is easy to prove that a decoding problem is definitely equivalent to an acquisition one. So, it is possible to design and run a MP algorithm on an *ad-hoc* graphical model to acquire the desired PN sequence. A large bibliography on MP algorithms is provided in literature, e.g., see [4, 1]. Therefore, we just remark the key parameters that are necessary to define and configure to correctly design these algorithms. They are listed below.

1. *Graphical Model* of the LFSR code, typically a loopy graph to get a lower complexity than that of a tree graph.
2. *Activation Schedule* of the algorithm, the order used to activate variable and check nodes, including the algorithm termination (e.g., the maximum number of iterations $I_{\max}$).
3. *Message-Passing Algorithm*, the algorithm used to compute and update soft information into the graph. Sum-Product (SP) or Min-Sum (MS) algorithms are the most common.

Concerning the graphical models, we refer to the construction shown in [1, 5]. Specifically, these models, called *redundant graphical models* (RGMs), are achieved by grouping a set of sub-graphs (*basic graphical models*, BGMs) each one univocally generated by one parity equation. All these equations have one common root that is the generating polynomial of the considered LFSR sequence, $P(D)$. In detail, exploiting the finite-fields property that $[P(D)]^{2^n} = P(D^{2^n})$ for a given n, we have an equivalent higher-degree generating polynomial $P_n(D) \triangleq P(D^{2^n}) \equiv P(D)$ that provides a correspondent parity equation. Thus, from each $P_n(D)$, the BGM $\mathbf{H}_n$ is built as follows

$$\mathbf{H}_n = \begin{pmatrix} g_{r_n} & \cdots & g_0 & 0 & \cdots & \cdots & 0 \\ 0 & g_{r_n} & \cdots & g_0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & g_{r_n} & \cdots & g_0 \end{pmatrix}_{N_r \times N_c} \quad (1)$$

where $N_r = M - r_n$ is the number of rows (or check nodes), $N_c = M$ is number of columns (or variable nodes), $r_n = \deg[P_n(D)]$, and M is the number of incoming observations. The RGM is got by grouping all BGMs

$$\mathbf{H_{RGM}} = \begin{pmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_{n_M} \end{pmatrix}_{N'_r \times N'_c}$$

where N'_r and N'_c are the number of rows and columns of $\mathbf{H_{RGM}}$, respectively, and $n_M = \lfloor \log_2 \{(M-1)/r_0\} \rfloor$ is the order of the higher-degree polynomial, with $r_0 = \deg[P(D)]$.

The next section introduces the *layered* activation schedule with a message-updating based on the generalized MS algorithm with the damping factor γ, [6].

3. MESSAGE-PASSING DETECTION UNIT

The architectural design of a generic *iterative Detection Unit* (iDU) [5] is depicted in Fig. 2. It is made up of an *input buffer* to collect M soft observations, an *iterative Processing Unit*, addressed to run iMP algorithms on loopy graphs, and a *Parity Control Unit*, which checks the parity at the end of each iteration. If the parity is satisfied a verification stage is run (correct/wrong detection is checked), otherwise a missed detection is got and a new acquisition try is run.

Focusing on the iterative Processing Unit, we resort to a very powerful and efficient re-formulation of the iMP algorithm based on the LDPC-like approach [7, 8], and known as *layered* or *shuffled* decoding. It is alternative to the traditional *flooding* schedule (FS), the difference lying in the way messages are propagated through decoding (activation schedule). The result is faster convergence, almost twice that of FS, both for codes with cycles and cycle-free. This is achieved through the early propagation of intermediate messages (in the form of *total information* or *soft-output*, *SO*) which are made available to the next elaborations immediately after their computation, instead of at the next iteration only.

The key idea is to view the code as the concatenation of smaller and independent sub-codes, or *layers*, which are decoded in a sequence, with the propagation, layer by layer, of *a-posteriori* probabilities. In the horizontal layered decoding (HLD), a layer is a subset of rows of the parity-check matrix $\mathbf{H}$; at the generic iteration t, HLD proceeds following the three steps below.

1. The total information of the connected bits y_n are used to derive the variable–to–check (*v2c*) messages involved in the update of CN m: $\mu^{(t)}_{m,n} = y_n - \epsilon^{(t)}_{m,n}$.
2. Check–to–variable (*c2v*) messages are updated separately on sign and magnitude:

$$\begin{cases} -\operatorname{sign}\left(\epsilon^{(t+1)}_{m,n}\right) = \prod_{j \in \mathcal{N}(m)\setminus n} -\operatorname{sign}\left(\mu^{(t)}_{m,j}\right) \\ \left|\epsilon^{(t+1)}_{m,n}\right| = \gamma \cdot \min_{j \in \mathcal{N}(m)\setminus n} \left\{\left|\mu^{(t)}_{m,j}\right|\right\} \end{cases} \quad (2)$$

3. The total information y_n of the involved bits are updated: $y_n = \mu^{(t)}_{m,n} + \epsilon^{(t+1)}_{m,n}$.

In (2), the min-sum (MS) update rule is used for magnitudes, and $\gamma \in \{0, 1\}$ is the *damping* factor, used to scale down *c2v* messages before propagation [6]. In this way, uncontrolled grows of messages and so possible drifts toward pseudo-codewords are prevented.

4. DECODER ARCHITECTURE

The HLD decoding algorithm has been implemented with a block-serial (or semi-parallel) decoder architecture based on serial processing units (SPUs). Although intrinsically slower, SPUs are preferable to parallel processing units (PPUs) for the reduced final complexity, the higher flexibility and the less congested routing of the chip. On the other hand, a fully parallel decoder based PPUs would result in extremely high complexity, [9].

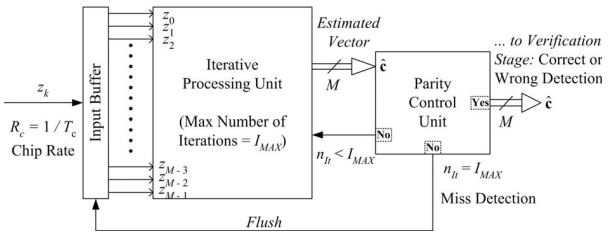


Fig. 2. Architectural design of an iDU.

4.1. Decoding Vectorization

Figure 3 shows the top-level decoder architecture. This is tailored to a m-sequence with only $q = 3$ non-null taps in the generating polynomial, which is put in the form $P_0(D) = 1 + D^a + D^b$, with $b > a > 1$; thus, the corresponding BGM $\mathbf{H_0}$ features three diagonals with offsets 0, $\alpha \doteq b-a$ and b. However, the proposed architecture can be easily modified to cope with generic *q-nomial* generators.

The rows of $\mathbf{H_0}$ spaced by $\beta \doteq b+1$ are then grouped in a layer to be updated in parallel. Thus, an array of SPUs with size $\Lambda = \lceil N_r/\beta \rceil = \lceil (M-b)/\beta \rceil$ is used to update (up to) Λ parity-check in parallel. The SPUs retrieve/store information from/to the *message* and the *soft-output* memory, respectively; also included in the *SO* data-path is a *shuffling* network, capable of shifting left or right the vector of data. To support multiple updates in parallel, memories are vectorized as well, and the data (*SO* and *c2v* messages) of Λ parity-check equations are packed in a single row of memory.

Figure 4 shows the internal organization of the vectorized *SO* memory, granting no hazard in the parallel access, and made of β rows with $\lceil M/\beta \rceil = \Lambda + 1$ elements each. As shown in Fig. 4, the data (*SOs*) packed on row i are relevant to the columns $i + k\beta$ of $\mathbf{H}_0$, with $k = 0, 1, ..., \Lambda$. This rule is fully valid for rows number 0 to $\lambda - 1 = \mathrm{rem}\,(M-b, \beta) - 1$, while rows λ to $\beta - 1$ are incomplete.

It turns out that this memory arrangement is also compatible with the decoding of next sub-codes $\mathbf{H}_i$, $i = 1, 2, ..., n_M$: by construction, rows spaced by β, can be still decoded in parallel with some differences. Let $N_{ri} = M - 2^i b$ be the number of parity-check constraints in sub-code i, then only $\Lambda_i = \lfloor N_{ri}/\beta \rfloor$ SPUs are working (we chose the least significant in the array), while the others remain unused. As already mentioned above, the last $\beta - \lambda_i$ layers of sub-code i, with $\lambda_i = \mathrm{rem}\left(N_{ri}, 2^i b\right)$, only need $\Lambda_i - 1$ SPUs.

In the processing of a generic layer, $q = 3$ arrays of *SOs* are taken from memory and updated; for the *l-th* layer of sub-code i these are the *SOs* with indices[1]: $[l, l+\beta, l+2\beta, ..., l+(\Lambda_i-1)\beta]$, $[l+\alpha_i, l+\alpha_i+\beta, l+\alpha_i+2\beta, ..., \ l+\alpha_i+(\Lambda_i-1)\beta]$, and $[l+b_i, l+b_i+\beta, l+b_i+2\beta, ..., \ l+b_i+(\Lambda_i-1)\beta]$, with $\alpha_i = 2^i(b-a)$ and $b_i = 2^i b$. If α_i is not an integer multiple of β, the three arrays of *SOs* above belong to different rows of memory, and no hazard arises in the HLD algorithm.

At any update, only part of a row is actually involved (Λ_i or $\Lambda_i - 1$ data), so that the *shifting* network of Fig. 3 is necessary to align data read from memory to the active SPUs (the Λ_i least significant ones in the array, as mentioned above). If the unused processors remain transparent to their inputs, messages not involved in the update are left untouched and are written back in memory. Note

[1]The following expressions hold only for $l < \lambda_i$, while for $\lambda_i \leq l < \beta$ the Λ_i*-th* processor remains unused and the arrays contain one less location.

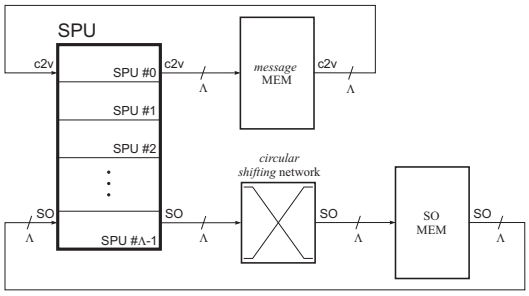


Fig. 3. Architecture of a layered iMP decoder for *m*-sequences.

	0	1	2	3	4	5	...	Λ-1	Λ
0	0	β	2β	3β	4β	5β	....	(Λ-1)β	Λβ
1	1	1+β	1+2β	1+3β	1+4β	1+5β	....	1+(Λ-1)β	1+Λβ
2	2	2+β	2+2β	2+3β	2+4β	2+5β	....	2+(Λ-1)β	2+Λβ
3	3	3+β	3+2β	3+3β	3+4β	3+5β	....	3+(Λ-1)β	3+Λβ
4	4	4+β	4+2β	4+3β	4+4β	4+5β	....	4+(Λ-1)β	4+Λβ
5	5	5+β	5+2β	5+3β	5+4β	5+5β	....	5+(Λ-1)β	5+Λβ
6	6	6+β	6+2β	6+3β	6+4β	6+5β	....	6+(Λ-1)β	6+Λβ
7	7	7+β	7+2β	7+3β	7+4β	7+5β	....	7+(Λ-1)β	7+Λβ
...	....	....+β	....+2β	....+3β	....+4β	....+5β	....	....+(Λ-1)β	....+Λβ
λ-2	λ-2	λ-2+β	λ-2+2β	λ-2+3β	λ-2+4β	λ-2+5β	....	λ-2+(Λ-1)β	λ-2+Λβ
λ-1	λ-1	λ-1+β	2λ-1	λ-1+3β	λ-1+4β	λ-1+5β	....	λ-1+(Λ-1)β	λ-1+Λβ
λ	λ	λ+β	λ+2β	λ+3β	λ+4β	λ+5β	....	λ+(Λ-1)β	
...	....	....+β	....+2β	....+3β	....+4β	....+5β	....	....+(Λ-1)β	
β-3	β-3	β-3+β	β-3+2β	β-3+3β	β-3+4β	β-3+5β	....	β-3+(Λ-1)β	
β-2	β-2	β-2+β	β-2+2β	β-2+3β	β-2+4β	β-2+5β	....	β-2+(Λ-1)β	
β-1	β-1	β-1+β	β-1+2β	β-1+3β	β-1+4β	β-1+5β	....	β-1+(Λ-1)β	

Fig. 4. Organization of the *SO* memory in a vectorized decoder.

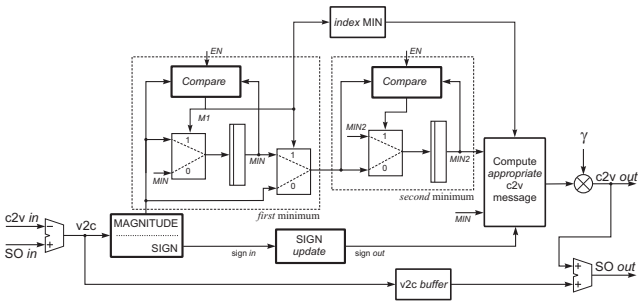


Fig. 5. Serial processing unit (SPU) architecture.

that only one single shifting network is implemented in the decoder, thanks to the use of *differential* cyclic shift techniques, where data are pre-rotated according to the next use. The shifting network is designed to circularly shift left or right an array of Λ data, with a maximum step of $2^{n_M} - 1$. An efficient implementation of this network may be based on a cascade of n_M stages of 2-to-1 multiplexers.

The architecture of a SPU is shown in Fig. 5. Check–to–variable messages are separately updated in sign and magnitude, and for magnitudes, the MS algorithm is serially implemented as follows.

i) First, the q input messages are reordered as to bring the least reliable message on the last position (*first* minimum).

ii) Then, the minimum of the remaining $q-1$ messages is computed (*second* minimum).

As a result, the absolute (*first*) and the *second* minimum of the q inputs are available. These are used along with the updated signs and the index of the *first* minimum to reconstruct the output *c2v* messages. Then, a shift-register with size $q+1 = 4$ is used to delay the input *v2c* message and update the *SO*.

Finally, note that instead of real multipliers, damping of *c2v* messages can be efficiently achieved with few *shift&add* networks.

4.2. Decoding Latency

Every sub-code of the *m*-sequence is decoded by the vectorized decoder with the same processing time, proportional to the numbers of layers β, the number of non-null taps q in the generator and the number of performed iterations $I_{\max}$. For the higher-order sub-codes, the number of messages to update is smaller, but as mentioned above, this only results in higher number of unused SPUs. Overall, the iMP decoding time can be expressed as:

$$T_{\text{iMP}} = \{I_{\max} \cdot (n_M + 1) \cdot q \cdot \beta + (q+4)\} \cdot t_{clk} \qquad (3)$$

with t_{clk} the clock period and $q + 4$ the latency, in clock cycles, of the data-path. As a result, the throughput of the iDU in number of acquisitions per time unit would approximately be: $\Gamma_{\text{iDU}} \simeq f_{clk}/(I_{\max} \cdot (n_M + 1) \cdot q)$.

5. PERFORMANCE & COMPARISONS

In this section the performance of an iterative Detection Unit (iDU) are compared to that of a full parallel (ML algorithm) and a simple serial algorithms. For the latter, the threshold was set to $\lambda = 0.85$, so to have a low false alarm and improve the acquisition time.

Referring to the communication system in Fig. 1, we assume to transmit the *m*-sequence generated with the 3-nomial $P(D) = 1 + D^7 + D^{18}$ (i.e., $a = 7$, $b = 18$ and $\alpha = 11$, $\beta = 19$), also referred to as g_{18} in the following, with period $N = 2^{18} - 1 = 262{,}143$. At the receiver side, the number of observations are $M = 1{,}024$ and a RGM with $n_M = 5$ is generated in agreement with the guidelines reported in Sect. 2. The iMP detector was implemented with $\Gamma = 53$ SPUs and with messages on 7 bits, while the damping factor γ was set to $6/16$ for optimal performance. In Fig. 6(a) it is evident that the wrong/missed detection probabilities of the iDU are higher than simple-serial false alarm, but in terms of correct detection (Fig. 6(b)) their probabilities are approximatively equivalent (there is a cross point). Nevertheless, in terms of acquisition times (as also shown in [1] and [5]) the iDU is definitely faster.

Comparing the full parallel to the iMP (Fig. 6(b)), the best performance (in terms of correct detection probability and acquisition time) is provided by the ML algorithm, but, as demonstrated in [5], the iMP acquisition time tends to that of the full parallel. Furthermore, the complexity of the iMP detector is clearly lower. Indeed, its complexity count is reported in Tab. 1 in terms of *adders*, *comparators*, *2-to-1 multiplexers* and *registers*. Considering that $1\,reg \simeq 1.5\,mux2$ and $1\,add \simeq 1\,comp \simeq 2\,mux2$, about 11,130 *mux2* are needed with messages on 7 bits. Finally, adding the 135,090 memory bits, and exploiting that $1\,bit \simeq 1\,mux2$, the iMP detector is memory-dominated, and its complexity is about 146,220 *mux2*.

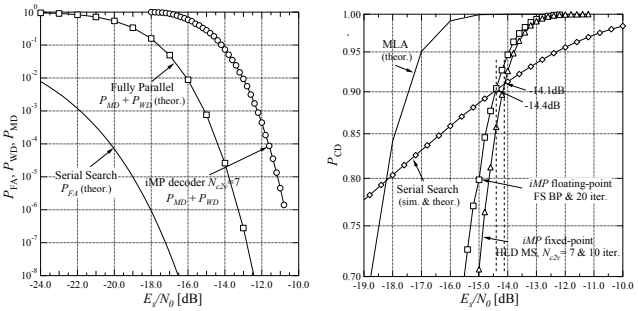
Conversely, the full parallel algorithm needs N branches performing serial correlations, each one using 1 *adder* and 2 *registers*. This is a clear under-estimation of the MLA complexity, because it neglects the logic for the selection of the best correlation (barrels of comparators) and memories. Nevertheless, it yields to about 9 millions *mux2* for a same 7-bit representation. Thus, the iMP complexity is about 2 orders of magnitude less than MLA.

6. CONCLUSION

This paper has presented a DS/SS acquisition technique based on iMP algorithm and performing a layered activation schedule on RGMs. The proposed implementation showed a very-fast acquisition time, which was achieved in a two-fold way: at the algorithmic level, the use of HLD with damping of the check–to–variable messages increased $2\times$ the average convergence speed; then, at the architectural level, the processing was vectorized and arranged in a cluster of SPUs. Comparing with other state–of–the–art techniques, the iMP detector performs a parallel acquisition with a complexity lower than that of a full parallel algorithm, and an acquisition time definitely shorter than that of a simple serial algorithm. This makes iDU with RGMs an effective solution to detect long spreading codes.

Table 1. iMP decoder complexity breakdown.

Unit	#	*add*	*comp*	*mux2*	*reg*
SPU	Λ	4	3	4	5+*q*
Shuff. Netw.	1	-	-	$n_m\Lambda$	-
RAM mem.	*bits* [row × col]				
c2v	$q(n_M+1)\beta \times 7\Lambda$	-	-	-	-
SO RAM	$\beta \times 8(\Lambda+1)$	-	-	-	-
total iMP	135,090 *bits*	212	159	212	424



(a) Missed/wrong detections. (b) Correct detection.

Fig. 6. Comparison between iMP, full parallel and serial detector.

7. REFERENCES

[1] O. W. Yeung and K. M. Chugg, "An iterative algorithm and low complexity hardware architecture for fast acquisition of long PN codes in UWB systems," *Springer J. VLSI and Signal Processing, Special Issue on UWB Systems*, vol. 43, no. 1, pp. 25–42, April 2006.

[2] L. Yang and L. Hanzo, "Acquisition of m-sequences using recursive soft sequential estimation," *IEEE Trans. Commun.*, vol. 52, no. 2, pp. 199–204, February 2004.

[3] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt, *Spread Spectrum Communications Handbook*. McGraw-Hill TELECOM, 2002.

[4] F. R. Kschichang, B. J. Frey, and H. A. Loeliger, "Factor graph and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[5] F. Principe, K. M. Chugg, and M. Luise, "Performance evaluation of message-passing-based algorithms for fast acquisition of spreading codes with application to satellite positioning," in *Proc. ESA Workshop on Satellite Navigation User Equipment Technologies NAVITEC 2006*. Noordwijk, The Netherlands: ESTEC, 11-13 December 2006.

[6] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity-check codes," *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406–414, March 2002.

[7] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Syst.*, vol. 11, no. 6, pp. 976–996, Dec 2003.

[8] D. Hocevar, "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes," in *IEEE Workshop on Signal Processing Systems, SISP 2004*, 2004, pp. 107–112.

[9] A. Blanksby and C. Howland, "A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar 2002.