An Iterative Algorithm and Low Complexity Hardware Architecture for Fast Acquisition of Long PN Codes in UWB Systems

On Wa Yeung and Keith M. Chugg

***Abstract*— Rapidly acquiring the code phase of the spreading sequence in an ultra-wideband system is a very difficult problem. In this paper, we present a new iterative algorithm and its hardware architecture in detail. Our algorithm is based on running iterative message passing algorithms on a standard graphical model augmented with multiple redundant models. Simulation results show that our new algorithm operates at lower signal to noise ratio than earlier works using iterative message passing algorithms. We also demonstrate an efficient hardware architecture for implementing the new algorithm. Specifically, the redundant models can be combined together so that memory usage can be reduced substantially. Our prototype achieves a combination of performance and complexity not possible with traditional approaches.**

I. Introduction

In an ultra-wideband (UWB) system, the source signal energy is spread over a bandwidth many times larger than its original bandwidth during transmission. As a result, the transmitted signal has a very low signal to noise ratio (SNR) and is completely buried in noise. Though this is a desirable property which minimizes interference to other users and makes it very difficult for an unintended receiver to detect and intercept the signal, it also presents the receiver designers with the very challenging problem of detecting and acquiring the signal at very low SNR.

Pseudo-random or pseudo noise (PN) sequences play an important role in a UWB system. They are periodic sequences with long period in practical systems. In a direct sequence ultra-wideband (DS/UWB) system, the transmitted signal is a train of very narrow pulses with polarities determined by the product of a PN binary sequence and the incoming binary source data sequence. For security reasons, it is often desirable to have PN sequences of very long period, so that to an unintended receiver over a short time interval, the sequences appear to be aperiodic and completely random [1], [2].

For a UWB receiver, the first step of demodulation is to de-spread the signal. In a DS/UWB system, this is achieved by multiplying the incoming samples by a local replica of the PN sequence. Therefore, the receiver must determine the unknown PN code phase embedded in the transmitted signal by analyzing the data collected from a short (compared to the PN code period) observation window so that it can synchronize the local replica. This is termed PN acquisition and will be the focus of this paper. Once the code phase is acquired, the receiver maintains the PN code synchronization through code tracking.

Traditionally, PN acquisition is achieved by searching explicitly over possible code phases. Reference signals corresponding to different code phases are correlated with the received signal and the one with the largest correlation is selected. For a DS/UWB system, the receiver estimates the arrival time of the pulses (i.e., the frame epoch), samples the incoming noisy signal and performs PN acquisition on these samples. The above process is repeated until acquisition is declared. Letting T_f be the pulse repetition period (frame time) and T_p be the pulse width, there are $\frac{T_f}{T_p}$ possible frame epoch values. In low data rate applications, typical values of the ratio $\frac{T_f}{T_p}$ range from 100 - 1000, thus the receiver has to perform up to 100-1000 PN acquisitions to locate the correct frame epoch. This number may be reduced if multi-path delay spread is exploited [3]. As explained in [4], [2], for a UWB system, the PN acquisition has to be completed quickly. If it is too slow, the correct code phase may never be acquired because the frame epoch may change due to timing drift before the receiver finishes evaluating the current frame epoch estimate. In this paper, we focus on fast PN acquisition and frame acquisition is not considered.

At one extreme of the traditional approaches to PN acquisition, all correlations are completed in one observation window. This is the full parallel search approach and it offers the best performance. At the other extreme, only one correlation is formed every observation window and acquisition is declared if a certain threshold is exceeded. This is the serial search approach. Hybrid search (i.e., correlating against only a subset of PN code phases in every observation window) is a compromise between the two extreme cases. Practically, parallel search is too expensive to implement for reasonably long sequences. As a result, serial and hybrid search are the only available options. As an example, the PN acquisition module of the UWB prototype in [5] was built using a hybrid approach to acquire a short PN sequence of period 128. However, for long PN sequences, serial search is often too slow and hybrid searches, at best, provides only a linear tradeoff between the speed and cost [6], [1].

Recently, iterative message passing algorithms (iMPAs) similar to the decoding algorithms for low density parity check codes (LDPC) and turbo codes were proposed in [4], [2] for fast PN acquisition in both direct sequence spread spectrum

The authors are with the Communication Science Institute, Department of Electrical Engineering, Viterbi School of Engineering, University of Southern California, Los Angeles, CA 90089-2565, {oyeung, chugg}@usc.edu.
This work was supported in part by the Army Research Office DAAD19-01-1-0477 and the National Science Foundation CCF-0428940.

(DS/SS) and DS/UWB systems. Similar approaches have also been proposed in [7], [8], [9]. Our exposition most closely follows that of [2]. These iterative algorithms offer the speed of parallel search and acquisition performance similar to that of serial search at short block lengths. Unlike parallel search, it is practical to implement the proposed algorithm in hardware to acquire PN sequences with long period. There are two drawbacks of the algorithm proposed in [4], [2]. First, the algorithm converges slowly at low SNR. Second, the performance of the algorithm does not scale well with observation length. Specifically, doubling the observation window length lowers the operating SNR of the traditional approaches by 3 dB but only by 1-2 dB for the proposed algorithm.

In this paper, we present a new improved iterative message passing algorithm and its hardware architecture based on the algorithm proposed in [2]. Specifically, we introduce multiple redundant models to mitigate the aforementioned drawbacks discussed in [4], [2]. The new algorithm converges faster and operates at lower SNR without increasing the hardware complexity. We will also demonstrate how to aggregate these multiples models into a single model to reduce memory usage. In our hardware prototype, the spreading sequence is of period $2^{22}-1$. Rapidly acquiring such a long sequence is impractical by both serial and parallel search, but the logic design based on our architecture can be easily fit into a small field programmable gate array (FPGA).

The remained of this paper is as follows. In Section II, we introduce the theory of operation. We then proceed to discuss various architectures for the main components of our module in Section III. Section IV gives a detailed account of our hardware implementations as well as various techniques we used in the optimization. Section V concludes the paper and gives directions for future work.

II. Theory of Operation

A. Maximal-length Sequences

A maximal-length sequence or m-Sequence is a linear feedback shift register (LFSR) sequence which has the maximum possible period for an r-stage shift register [10]. As its name implies, an m-Sequence x_k can be generated by an r-stage linear feedback shift register structure as shown in Fig. 1. When the registers are loaded with any non-zero values, the generated sequence will cycle through all 2^r-1 possible non-zero states before repeating (i.e., its period is 2^r-1). Mathematically, the sequence structure can be expressed as

$$x_k = g_1 x_{k-1} \oplus g_2 x_{k-2} \oplus ... \oplus g_r x_{k-r} \tag{1}$$

where $g_0 = g_r = 1$, $g_k \in \{0,1\}$ for $1 < k < r$ and $\oplus$ is the modulo-2 addition. The generator polynomial is $g(D) = D^r + g_{r-1}D^{r-1} + g_{r-2}D^{r-2} + ... + D^0$ where D is the unit delay operator [10]. Given r, there is only a very limited set of g_k values that generates an m-Sequence. Because of their excellent auto-correlation and cross-correlation properties, m-Sequences are widely used as spreading sequences in spread spectrum systems [10], [1].

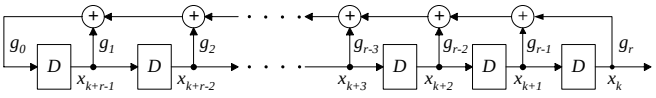


Fig. 1. Linear feedback shift register (LFSR) structure for m-Sequence generation.

B. Signal Model

For a DS/UWB system, a standard model for acquisition characterization is [2], [1]

$$z_k = \sqrt{E_c}(-1)^{x_k} + n_k \tag{2}$$

where z_k, $0 \le k \le M-1$, is the noisy sample received by the acquisition module, x_k, $0 \le k \le M-1$, is the spreading m-Sequence, E_c is the transmitted energy per pulse and n_k is additive white Gaussian noise (AWGN) with variance $\frac{N_0}{2}$. We also assume that x_k is generated by an r-stage LFSR and $r \ll M \ll 2^r - 1$. This is a much simplified model which assumes no data modulation and does not include the effect of jamming, oversampling, etc, but it is widely used in the literature to benchmark the performance of PN acquisition algorithms.[1]

The goal of the acquisition module is to estimate x_k based on z_k $0 \le k \le M-1$ for a given frame epoch estimate and decide whether the frame epoch estimate is correct. In our design, we obtain the estimate of x_k, denoted by $\hat{x}_k$, by running an iterative message passing algorithm. Because $\hat{x}_k$ has to be consistent with (1), once r consecutive $\hat{x}_k$ are obtained, the rest of the sequence is determined by extrapolating the estimate by (1). As the last step, z_k is correlated with $\hat{x}_k$, $0 \le k \le M-1$ to check whether the correlation threshold is reached.

C. Iterative Message Passing Algorithm (iMPA) for Fast PN Acquisition

In traditional PN acquisition approaches, the received sequence z_k is correlated with up to 2^r-1 PN sequences generated by different x_0, x_1, ..., x_{r-1} combinations for the whole observation window and the algorithm chooses the phase corresponding to the highest correlation. In terms of computation complexity, the main difference between parallel search and serial/hybrid search is whether all correlations are completed every observation window. Since each correlation requires $M-1$ additions, the computation complexity is of $O(M \cdot 2^r)$ for all of these traditional approaches. For parallel search, all valid configurations of x_k are correlated, we can therefore interpret it as maximum likelihood (ML) decoding of x_k from (2) and serial/hybrid search as approximations to ML decoding. Based on this observation, we formulate the PN acquisition problem as a decoding problem and apply an iterative message passing algorithm similar to turbo code [11], [12] or LDPC decoding [13], [14], [15].

[1]As shown in [2], this model and the algorithms developed in this paper can be modified to work in sinusoidal carrier systems such as direct sequence spread spectrum systems (DS/SS). In such systems, the model of (2) should be generalized to account for an unknown carrier phase, θ_c. The approach suggested in [2] is to search over a finite set of carrier phase values.

Since the inception of turbo codes, iterative message passing algorithms have been widely studied. They can be easily derived by constructing the corresponding graphical models for the system and applying a standard set of rules. It is well understood that if the graphical model has no cycles, the algorithm is equivalent to maximum likelihood decoding. Otherwise, the algorithm is heuristic and sub-optimal [16], [17], [18], [19], [20], [21]. However, it is often a good approximation to maximum likelihood decoding and offers near-optimal performance as in the case of turbo code and LDPC decoding.

In practical applications, cyclic graphical models are chosen for low complexity decoding. The graphical models chosen have a significant impact on the performance of the algorithm. Heuristically, a good model should neither have short nor regular cycles [16], [22], [23]. Several graphs corresponding to the same generator polynomial $g(D) = D^{15} + D^1 + D^0$ are shown in [2, Fig. 2] and each implies a different decoding algorithm.

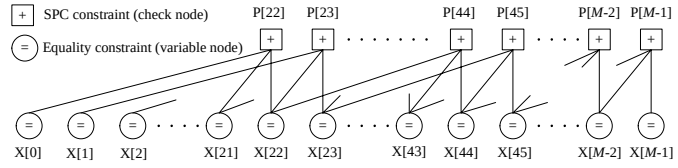
For a binary variable X, the message passed (i.e., soft information) in a cyclic graph is an approximation of the negative log-likelihood ratio $-\log \frac{Pr(X=1)}{Pr(X=0)}$ [21]. In our case, in each iteration, the algorithm successively updates messages and decisions are made by comparing a decision message M_{dec} to 0 where M_{dec} is an approximation of $-\log \frac{\Pr(x_k=1)}{\Pr(x_k=0)}$. If $\mathrm{M}_{dec} \geq 0$, $\hat{x}_k = 0$, otherwise, $\hat{x}_k = 1$. The absolute value of M_{dec} can be interpreted as the confidence of the decision. If the algorithm converges, M_{dec} will stabilize after certain number of iterations indicating some level of confidence in the decisions.

A detailed discussion of iterative message passing algorithms is beyond the scope of this paper. In the remaining sections, we consider acquiring the m-Sequence with generator polynomial $g(D) = D^{22} + D^1 + D^0$ and only the details relevant to our example are presented. Interested readers can refer to [21], [24], [16], [11], [17], [25] for further details.

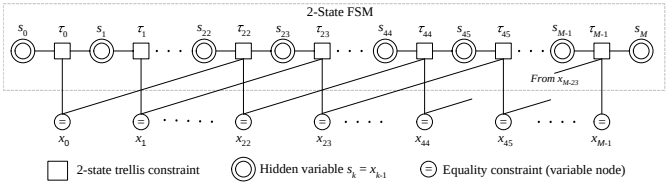
Similar to the polynomial $g(D) = D^{15} + D^1 + D^0$ presented in [2], our polynomial can also be represented by several graphical models with two shown in Fig. 2(a) and 2(b). Decoding algorithms based on both models offer similar performance as in [2] and suffer the same problems. The slow convergence experienced by the algorithms is similar to that of LDPC decoding and can be attributed to the weak constraints and the flooding activation schedule. The SNR scaling problem is attributed to the existence of regular cycle structures in the graphs in [2]. Qualitatively speaking, this is a "bad" graphical model to apply standard iterative message passing algorithm. The problem is tackled in [2] by inverting the signs of the set of messages corresponding to the least reliable decisions and rerunning the algorithm if acquisition fails. This approach does improve sensitivity, but it still requires many iterations. This motivates us to find a better graphical model on which we can apply the standard iterative message passing algorithm and is more amenable to hardware implementation.

D. Graphical Models with Redundancy

To improve the performance of the iMPA, we introduce a new decoding graph for $g(D) = D^{22} + D^1 + D^0$. It is



(a) Tanner graph for $g(D) = D^{22} + D^1 + D^0$.



(b) Tanner-Wiberg graph for $g(D) = D^{22} + D^1 + D^0$ with hidden variable $S_k = x_{k-1}$ introduced.

Fig. 2. Different graphical models for $g(D) = D^{22} + D^1 + D^0$.

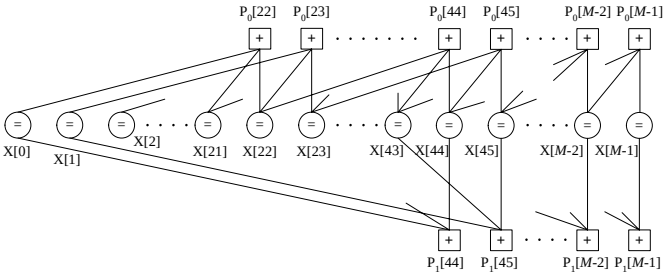


Fig. 3. Forming the 1^{st} order graphical model using the primary model ($g(D) = D^{22} + D^1 + D^0$) and the 1^{st} order auxiliary model ($g(D) = D^{44} + D^2 + D^0$).

constructed using multiple graphical models each of which fully captures the PN code structure. In this sense, the model has redundancy. This is equivalent to adding redundant parity checks to the standard parity check matrix. The technique is also applied in soft decoding of some of the classical codes [26], [27], [28]. Fig. 3 shows the special case of using two models. Each of the subgraphs is based on a different generator polynomial to the same m-Sequence. Mathematically, we introduce different non-primitive polynomials to generate the same sequence. For example, let x_k be the sequence generated by $g(D) = D^{22} + D^1 + D^0$, we have the following equations:

$$x_k \oplus x_{k-1} \oplus x_{k-22} = 0 \quad (3)$$

$$x_{k-1} \oplus x_{k-2} \oplus x_{k-23} = 0 \quad (4)$$

$$x_{k-22} \oplus x_{k-23} \oplus x_{k-44} = 0 \quad (5)$$

Adding (3), (4) and (5) together, we have $x_k + x_{k-2} + x_{k-44} = 0$. Therefore, $g(D) = D^{44} + D^2 + D^0$ also generates the same sequence. The argument can be easily extended to show that

$$g(D) = D^{22 \cdot 2^n} + D^{2^n} + D^0,\ n = 0, 1, 2, 3... \quad (6)$$

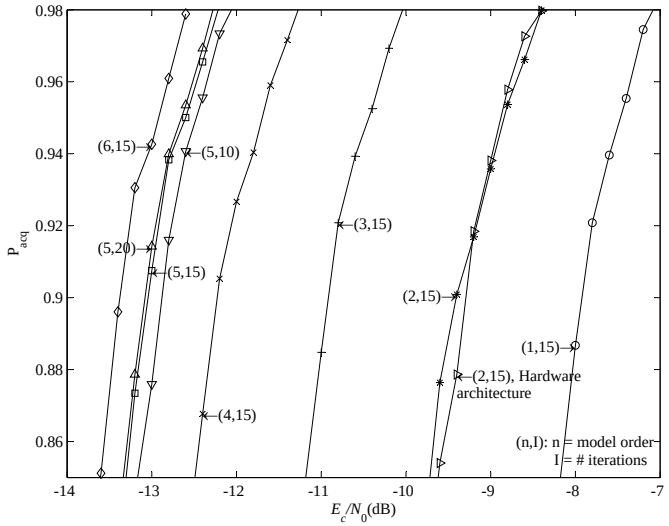


Fig. 4. Acquisition performance vs. E_c/N_0 for using an n^{th} order model on $g(D) = D^{22} + D^1 + D^0$, $M = 1024$. The acquisition performance of our hardware implementation (see Section IV) is marked as "(2,15), hardware architecture".

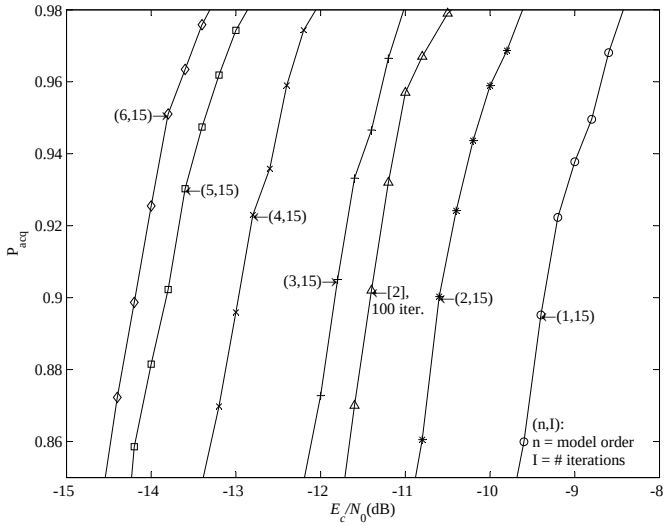


Fig. 5. Acquisition performance vs. E_c/N_0 for using an n^{th} order model on $g(D) = D^{15} + D^1 + D^0$, $M = 1024$. As a reference, the acquisition performance by running the [2] algorithm is marked as " [2], 100 iter.".

all generate the same sequence. In this paper, we refer the graphical model based on (6) as the n^{th} order auxiliary model and the one based on $g(D) = D^{22} + D^1 + D^0$ as the primary model. Also, we refer to the model that combines the primary model and the 1^{st}, 2^{nd} ... $(n-1)^{th}$ order auxiliary models as the n^{th} order model. Our decoding graph for an n^{th} order model is formed by constraining the output of primary model and each of the i^{th} order auxiliary models $1 \leq i \leq n$ to be equal. As an example, the graph of the 2^{nd} order model is shown in Fig. 3. The performance improvement by combining multiple models is shown in Fig. 4. Even though each individual auxiliary model produces very unreliable decoding decisions, combining them improves the convergence behaviour dramatically. We gain around 1 dB gain for each additional auxiliary model introduced. Only 10 iterations are required for practical convergence for a 5^{th} order model. Our multiple model algorithm also works for other m-Sequences. As a comparison, Fig. (5) shows the performance for $g(D) = D^{15} + D^1 + D^0$ where the curve for algorithm in [2] is also included.

Our baseline algorithm is summarized in Algorithm 1. The complexity of both the decoding and correlation operations is of $O(M)$, therefore our algorithm is also of $O(M)$ complexity. There is substantial complexity reduction compared to the traditional approaches. Also, our new algorithm offers better performance with no additional complexity compared to the approach in [2] since we reduce the number of iterations dramatically.

Input: $\mathrm{M}_{ch}[k] = z_k$, $0 \leq k \leq M-1$;
Output: Acquisition decision $\hat{x}_k$, $0 \leq k \leq M-1$;
for *i=1..I* **do**
  run the iterative message passing algorithm to get M_{dec}, the algorithm can be based on different graphical models such as Fig. 3 and Fig. 10(c);
  if $\mathrm{M}_{dec} \geq 0$ **then**
    $\hat{x}_k = 0$
  else
    $\hat{x}_k = 1$
  end
  divide $\{\hat{x}_k\}$ into non-overlapping 22-pulse segments;
  choose the segment corresponding to the maximum $\sum_{k=22\cdot j}^{22\cdot j+21} |\mathrm{M}_{dec}[k]|$;
  set $\hat{x}_k$ equal to the extrapolated value of the chosen segement by (1), $0 \leq k \leq M-1$;
  correlate z_k with $\hat{x}_k$: $c = \sum_{k=0}^{k=M-1} z_k \hat{x}_k$;
  if *c > threshold* **then**
    declare acquisition;
    break;
  end
end

Algorithm 1: Baseline iMPA algorithm for fast PN acquisition.

III. High Level Design for the Iterative Decoder

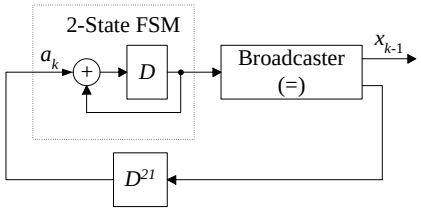
In this section, we present the hardware architecture of the basic building blocks in our iMPA algorithm. Assuming using an n^{th} order model, the pulses are decoded by n different models during each iteration. The hardware module that performs the iterative message passing algorithm for each auxiliary model is an iterative decoder.

A. Forward Backward Algorithm Based Iterative Decoder

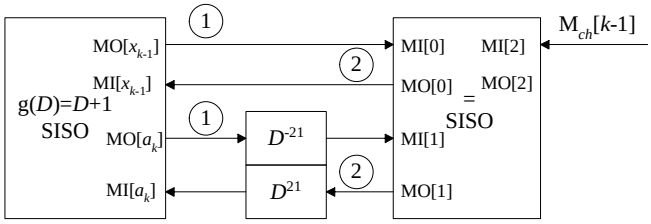
The basic building block in our algorithm is an iterative decoder that decodes the sequence generated by $g(D) = D^{22}+D^1+D^0$. We have two hardware architecture candidates: one based on Fig. 2(a) and another based on Fig. 2(b). Simulation shows that both architectures performs similarly (the difference in sensitivity is less than 0.3 dB).

If we choose the Tanner graph representation as shown in Fig. 2(a), the number of messages needed to be saved in each iteration is the number of edges in the graph. Therefore, the minimum storage requirement is $3M$ messages.

Alternatively, if we base our decoder on a Tanner-Wiberg



(a) Decomposition of $g(D) = D^{22}+D^1+D^0$ as a combination of a 2-state FSM, a broadcaster and a delay block.



(b) Corresponding $g(D) = D^{22}+D^1+D^0$ iterative decoder architecture, the circled number is the activation order.

Fig. 6. Deriving the $g(D) = D^{22} + D^1 + D^0$ decoder architecture from the LFSR structure.

graph [16] with hidden variables introduced as shown in Fig. 2(b), we have a more memory efficient hardware architecture. Equations similar to [2, (23)] to [2, (29)] can readily be obtained from this graph. This graph is an explicit index diagram [25], [21]. For readers not familiar with Tanner-Wiberg graphs, the update equations may be more easily explained by considering Fig. 6(a) which decomposes the sequence generating LFSR structure into three parts: one 2-state $g(D) = D + 1$ finite state machine (FSM)[2], one delay block (D^{21}) and one broadcaster (i.e., an equality constraint). Applying the standard iterative message passing rules [25], [21], we derive the decoding graph (Fig. 6(b)) by replacing each component by a soft-in soft-out (SISO) module which performs the a-posteriori probability (APP) decoding [29].

The relationship between Fig. 6(b) and Fig. 2(b) is that Fig. 2(b) is an explicit index diagram and Fig. 6(b) is an implicit index diagram where the time index is hidden in the graphical representation [21], [25]. The associated iterative processing is the same in both cases.

Let $\mathrm{MI}[i]$ and $\mathrm{MO}[i]$ be the input and output messages with ports defined in Fig. 6(b). The broadcaster SISO update equation is $\mathrm{MO}[i] = \sum_{j=0}^{j=2} \mathrm{MI}[j] - \mathrm{MI}[i]$ ([21], [25]).

The 2-state $g(D) = D + 1$ recursive FSM SISO can be implemented by the forward backward algorithm (FBA) [21]. Let F_k and B_k be the forward and backward state metric and $\mathrm{MI}[x_k]$, $\mathrm{MO}[x_k]$, $\mathrm{MI}[a_k]$ and $\mathrm{MO}[a_k]$ be the input and output ports defined in Fig. 6(b). The update equations for each iteration are:

$$\mathrm{F}_0 = 0 \tag{7}$$

$$\mathrm{B}_M = 0 \tag{8}$$

$$\begin{aligned}\mathrm{F}_{k+1} = &\min(\mathrm{MI}[a_k],\ \mathrm{F}_k) - \\ &\min(0,\ \mathrm{F}_k + \mathrm{MI}[a_k]) + \mathrm{MI}[x_k]\end{aligned} \tag{9}$$

$$\begin{aligned}\mathrm{B}_k = &\min(\mathrm{MI}[a_k],\ \mathrm{MI}[x_k] + \mathrm{B}_{k+1}) - \\ &\min(0,\ \mathrm{B}_{k+1} + \mathrm{MI}[x_k] + \mathrm{MI}[a_k])\end{aligned} \tag{10}$$

$$\begin{aligned}\mathrm{MO}[a_k] = &\min(\mathrm{B}_{k+1} + \mathrm{MI}[x_k],\ \mathrm{F}_k) - \\ &\min(0,\ \mathrm{F}_k + \mathrm{B}_{k+1} + \mathrm{MI}[x_k])\end{aligned} \tag{11}$$

$$\mathrm{MO}[x_k] = \mathrm{F}_{k+1} + \mathrm{B}_{k+1} - \mathrm{MI}[x_k] \tag{12}$$

$$\mathrm{M}_{dec} = \mathrm{MI}[x_k] + \mathrm{MO}[x_k] \tag{13}$$

From the above equations, we can see that the FSM SISO requires two types of memory. The first type is for storing the $2M$ messages passed between the $g(D) = D + 1$ SISO and the broadcaster SISO. Their values are updated based on the results from the previous iteration. The second type is for storing the FSM state metrics F_k and B_k, which are recalculated during every iteration. In other words, the FSM state metric memory can be reused once operations in the current iteration are finished. Therefore, we do not need to store B_k if $\mathrm{MO}[\cdot]$ are updated immediately once both F_k and B_{k+1} become available. In Section IV, we will show that the state metric memory can be reduced substantially by updating the state metrics segment by segment to reuse the memory within the current iteration. If the segment size is $M/8$, the total memory requirement becomes $M/8$ state metrics + $2M$ messages which is substantially less than the $3M$ messages requirement based on 2(a). For low data rate applications, the transistor count for our circuit is dominated by memory instead of logic. Therefore, the architecture shown in Fig. 6(b) is preferred because of its lower memory usage.

In Fig. 6(b), we show one type of activation schedule, the 2-state FSM SISO completes the message update, sends them to the broadcaster, then the broadcaster updates and returns the messages. This completes one iteration.

B. Forming an n^{th} Order Decoder

Once we have all the auxiliary model decoders ready, forming an n^{th} order model decoder is straightforward. We only need to form an additional broadcaster (equality) constraint and the decoding architecture follows directly by applying the standard iterative message passing rules as shown in Fig. 7.

If we only consider a 2^{nd} order model and choose the SISO structure to be of the type Fig. 2(a), then Fig. 7 is equivalent to Fig. 3. The memory requirement equals to $6M$ messages which is the sum of the memory requirement for each SISO.

C. Simplification of an Auxiliary Model Decoder Using Index Partitioning

An advantage of using auxiliary models as defined in (6) is that all auxiliary decoders can be constructed using the $g(D) = D^{22} + D^1 + D^0$ decoder. This is achieved by index partitioning on the output of the higher order model.

[2]Note that this is the feedback polynomial. Viewed as a code, the generator is $1/(1 + D)$, which is an accumulator.

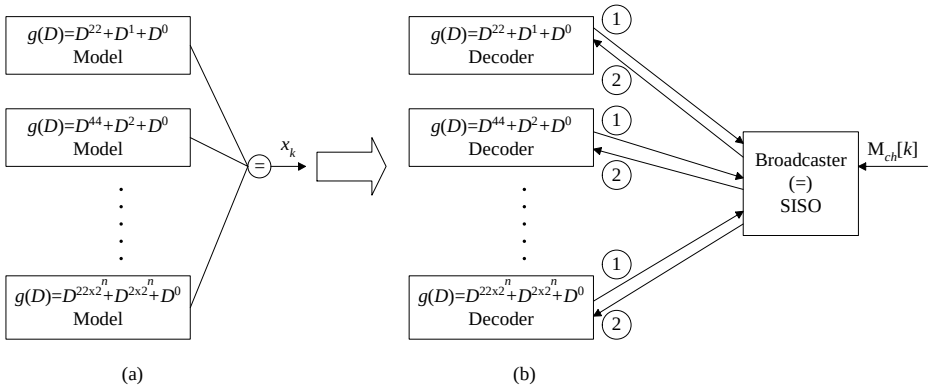


Fig. 7. Iterative decoder architectures for an n^{th} order model: (a) is the combined model; (b) is the iterative decoder architecture with the activation order circled.

Specifically, the index partitioned output is equivalent to the output of the primary model.

As an example, consider the FSM that generates the $g(D) = D^{44} + D^2 + D^0$ sequence. As shown in Fig. 8(a), we can model this as two identical FSMs each generating the $x_k = x_{k-1} + x_{k-22}$ sequence. One generates the sequence at odd indices and the other generates the sequence at even indices. The corresponding decoder is shown in Fig. 8(b) which consists of two $g(D) = D^{22} + D^1 + D^0$ decoders. In this case, since each decoder only decodes $M/2$ pulses, the total memory requirement for messages is the same as an M-pulse $g(D) = D^{22} + D^1 + D^0$ decoder.

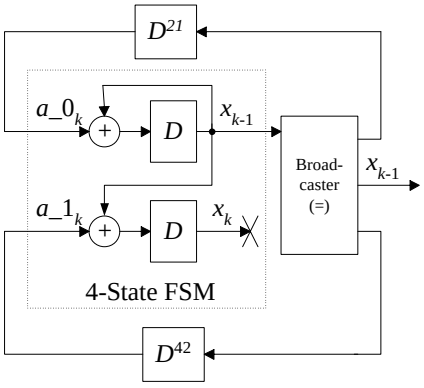
This idea extends to higher order auxiliary model decoders. Specifically, x_k generated by (6) can be partitioned into 2^n sub-sequences: $x_{2^n k+i}$, $0 \le i \le 2^n - 1$ with each sub-sequence generated by $g(D) = D^{22} + D^1 + D^0$. The corresponding decoder can be constructed using multiple $g(D) = D^{22} + D^1 + D^0$ decoders similar to Fig. 8(b).

IV. Hardware Architecture

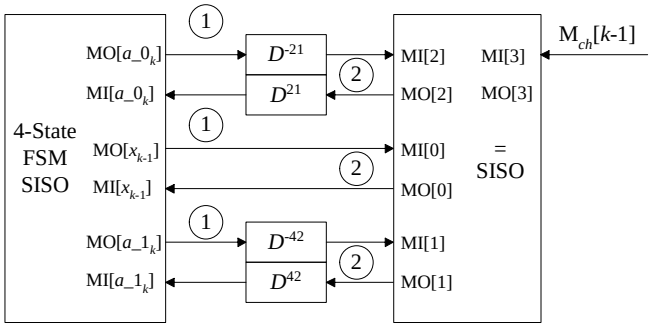
In this section, we consider the case of decoding the PN sequence $g(D) = D^{22} + D^1 + D^0$ over an observation window of $M = 1024$ using the 2^{nd} order model architecture. The block diagram of our acquisition module is shown in Fig. 9.

A. 4-State FSM Decoder

As shown in Fig. 9, instead of using Fig. 7 as our PN estimator architecture which has three 2-state FSM SISOs (one for $g(D) = D^{22} + D^1 + D^0$ and two for $g(D) = D^{44} + D^2 + D^0$), we combine the two models together using a single 4-state FSM as shown in Fig. 10(a). The new FSM captures all the information of the original FSMs and lowers the memory requirements from $4M$ messages plus state metrics to approximately $3M$ messages plus state metrics as demonstrated below. Moreover, by using a single FSM, we save routing resources by lowering the bandwidth requirement for the channel metrics ($\mathrm{M}_{ch}[k] = z_k$) memory since it is now accessed only by one FSM-SISO instead of three FSM SISOs. Using the 4-state FSM does require more logic in the FSM SISO implementation, but this increase is justified by the the additional savings in memory and routing.



(a) Decomposition of a $g(D) = D^{44} + D^2 + D^0$ FSM into two $g(D) = D^{22} + D^1 + D^0$ FSMs by index partitioning.



(b) $g(D) = D^{44} + D^2 + D^0$ decoder architecture.

Fig. 8. Implementing a $g(D) = D^{44} + D^2 + D^0$ decoder using two $g(D) = D^{22} + D^1 + D^0$ decoders.

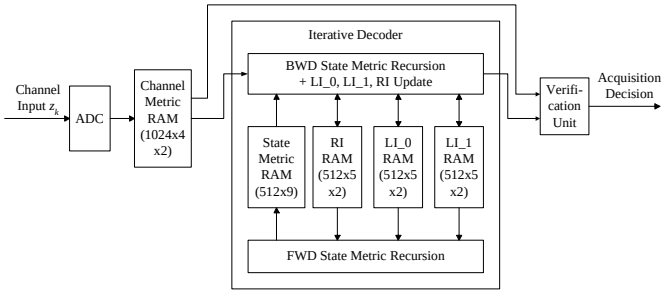


Fig. 9. Block diagram of the acquisition module for $g(D) = D^{22}+D^1+D^0$.

TABLE I
STATE TRANSITION TABLE OF THE 4-STATE FSM.

$S_{k-1} = \{x_{k-2}, x_{k-1}\}$	S_k	x_k	x_{k-22}	x_{k-44}
00 (0)	00 (0)	0	0	0
00 (0)	01 (1)	1	1	1
01 (1)	10 (2)	0	1	0
01 (1)	11 (3)	1	0	1
10 (2)	00 (0)	0	0	1
10 (2)	01 (1)	1	1	0
11 (3)	10 (2)	0	1	1
11 (3)	11 (3)	1	0	0

Our 4-state FSM decoder is also based on the forward backward algorithm. We define the state as $S_k = \{x_{k-1}, x_k\}$ and the corresponding decoder is shown in Fig. 10(b). Again, this is an implicit index digram. The explicit index diagram (i.e., the Tanner-Wiberg graph) is shown in Fig. 10(c). The state transition table is shown in Table I and the messages passed are shown in detail in Fig. 10(d).

The update equations are obtained by applying the standard message passing rules [21] on either Fig. 10(b) or Fig. 10(c). They are listed from (14) to (30) in the appendix.

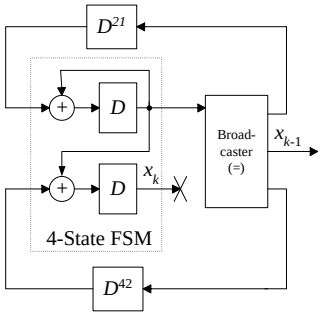
Simulation results shows that the 4-state FSM decoder implementation improves the performance by 0.2 dB in $\frac{E_c}{N_0}$ as compared to the three 2-state FSM implementation.

We can continue to combine multiple auxiliary models to form a single FSM. For example, we can implement a 3^{rd} order model using a 16-state FSM. However, the exponential growth in state metric memory may outweigh any savings in the message memory for larger n.

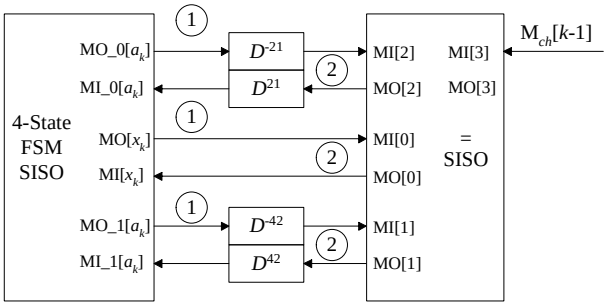
B. Forward Backward Algorithm Architecture for Multiple Index Segments

To reduce the internal FSM state metric memory, we divide the observation window into multiple segments and run the forward backward algorithm (FBA) segment by segment. This is a standard approach for implementing the Viterbi and turbo decoders [30], [31].

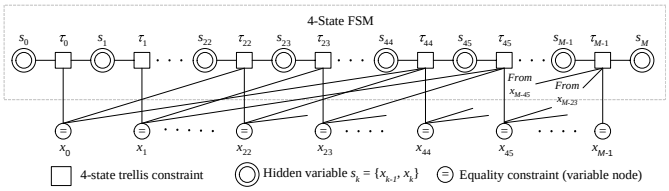
In our prototype system, we divide the observation window (1024) into 8 segments. There is one forward unit and one backward unit running 15 iterations. During each iteration, the forward unit updates the state metric sequentially from pulse 0 to 1023. The backward unit computes the state metric in the following order: 127→0, 255→128, ..., 1023→896. Such a sequence of calculations results in one problem: we



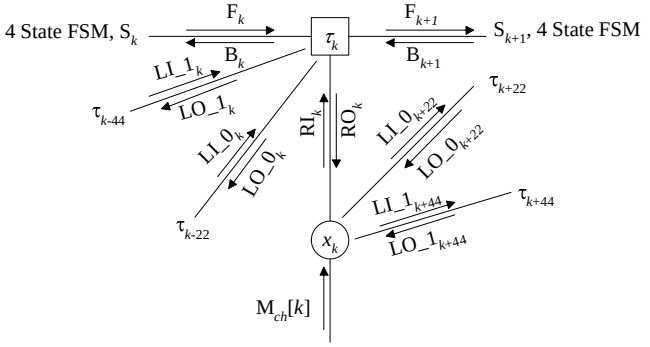
(a) Combining the encoders for the primary model and the 1^{st} order auxiliary model to form a 4-state FSM encoder. The output x_k is redundant.



(b) Corresponding iterative decoder for the 4-state FSM encoder. The circled number is the activation order. This is an implicit index diagram.



(c) Tanner-Wiberg graph for the 4-state FSM. This is an explicit index diagram.



(d) Detailed view of the messages passed in and out of the 4-state FSM trellis constraint node. For specific update equations, see (14) - (30).

Fig. 10. Implicit and explicit index diagrams for the 4-state FSM decoder of $g(D) = D^{22} + D^1 + D^0$.

do not know the backward metric $\mathrm{B}_{128}[i]$, $0 \le i \le 3$ when computing $127 \rightarrow 0$, $\mathrm{B}_{256}[i]$, $0 \le i \le 3$ when computing $255 \rightarrow 128$, etc. The problem is solved in [30] and [31] by running the backward unit for an additional "warm-up" period. The approach is motivated by the fact that the backward state metric at the segment boundary can be well approximated by starting a backward state recursion just several constraint lengths away. Excluding the warm-up, (i.e., setting $\mathrm{B}_{128}[i] = 0$) will incur a loss of around 0.25 dB in E_c/N_0. To run a design using the warm-up approach at full-speed, an additional backward unit is required so that one unit warms up while the other is doing the update [30], [31]. The additional unit can be saved if we do not use the warm-up approach but instead copy the $\mathrm{B}_{128}[i]$ values from the previous iteration. This is feasible because the warm-up period is only required if we are trying to approximate an FBA-SISO in isolation. For an iterative system, starting the backward recursions based on earlier iteration values is equivalent to a change in the activation schedule for the iMPA on the cyclic graph, and as such does not significantly affect the performance. This is a known architecture for implementing iterative decoders with forward-backward based SISO decoders (e.g., see [32], [33], [34]). Once both the forward and backward state metrics become available, $\mathrm{LI\_0}_k$, $\mathrm{LI\_1}_k$, RI_k and $\mathrm{M}_{dec}[k]$ are computed and the FSM state metric memory is released immediately. The processing pipeline is shown in Fig. 11 which shows the update sequence as well as the corresponding memory access.

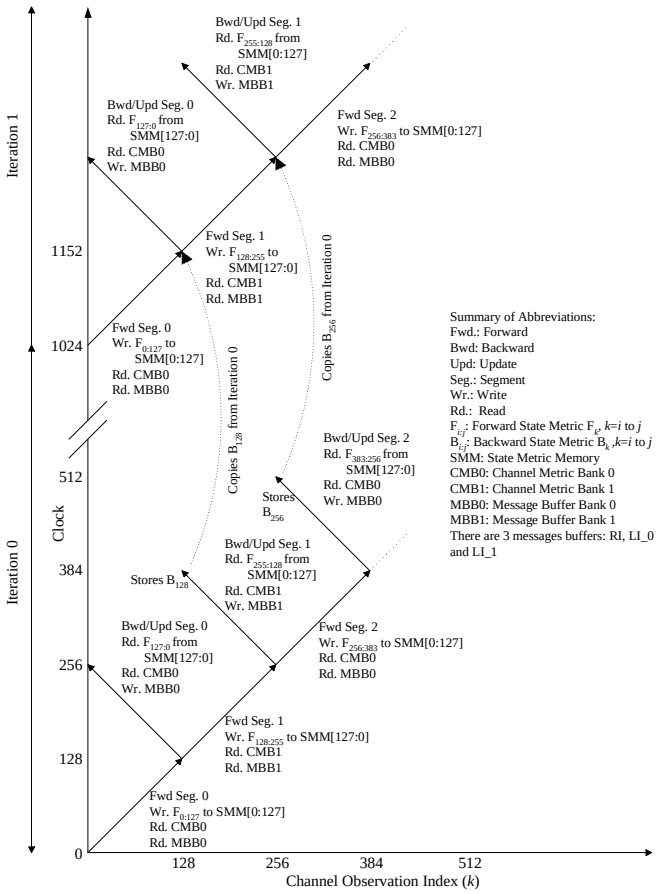


Fig. 11. Processing and memory access pipeline for the 4-state decoder.

C. Bit Width

The bit widths in our system are determined by simulations in two steps. First, we fixed $\mathrm{LI\_0}_k$, $\mathrm{LI\_1}_k$, RI_k to be of 16 bits and determine that 4 bits of ADC output is sufficient. Compared to floating point, there is a performance loss of only 0.2 dB.

The performance for various bit width combinations is shown in Fig. 12. For each ADC bit width, we have optimized the scale q that sets the ADC dynamic range (ADC_{out} = quantize($q \cdot z_k$)) for performance. For a 4-bit ADC, q_{opt} is found to be 1.65 by simulation. As a reference, we also show the performance for the standard mid-point loading $q = 3.5$ when the ADC is of 4 bits.

The second step is to determine the bit width for the messages $\mathrm{LI\_0}$, $\mathrm{LI\_1}$ and RI. This is necessary since their values may grow as the decoder iterates. To avoid using excessive bits for storage, we have to clip them after each (FBA/=) SISO activation. As shown in Fig. 12, 5 bits are sufficient for our application when the ADC bit width is 4.

To determine the bit width for the state metric, we rely on the fact that for a given k, we are only interested in the difference between $\mathrm{F}_k[i]$ $0 \le i \le 3$, not their values. Therefore, we only need the bit width to be big enough for the differences. If we subtract $\mathrm{F}_k[0]$ from $\mathrm{F}_k[i]$ $0 \le i \le 3$, the differences (i.e., the normalized $\mathrm{F}_k[i]$) can be shown to be bounded between -128 to 127 for 5-bit messages by an argument similar to the ones in [35] and [21]. As a result, it can be represented by 8 bits. Similarly, the normalized $\mathrm{B}_k[i]$

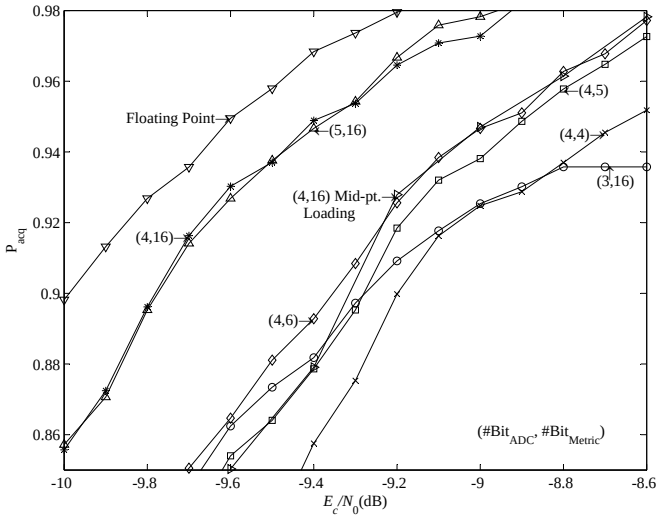


Fig. 12. P_{acq} vs. E_c/N_0 for different bit width combinations, $g(D) = D^{22} + D^1 + D^0$.

$0 \leq i \leq 3$ can be represented by 8 bits for $0 \leq k \leq 1024$. Since the normalized $\mathrm{F}_k[0]$ and $\mathrm{B}_k[0]$ are always 0, we do not need to store them. The normalization approach is what we apply for all binary variables because it reduces memory usage by half and only requires one subtraction. For example, $\mathrm{LI\_0}_k$ is a shorthand for $\mathrm{LI\_0}_k[1]$ where $\mathrm{LI\_0}_k[0] = 0 \ \forall \ k$. For the 4-ary state metric variables $\mathrm{F}_k[i]$ and $\mathrm{B}_k[i]$, the normalization approach is less attractive. It only reduces the memory usage by one-fourth at the expense of impacting the frequency scaling of our circuit by requiring three additional subtractions in the critical path of the forward and backward recursions. As a result, we do not perform normalization and use 9 bits to represent the state metric instead of 8 bits. This additional 1-bit approach is commonly used in Viterbi decoders and is proven to be correct with two's complement arithmetic [35], [21].

D. Partitioning the Memory into Banks

In our prototype design, we have several modules concurrently accessing memory. By carefully partitioning the memory, contention can be avoided without the use of multiport memory. The access pipeline for the above memories is shown in Fig. 11.

For the message memories (LI_0, LI_1 and RI), we divide them into two banks of 512 entries. One bank is for the odd FBA segment and the other for the even FBA segment. By this arrangement, there are at most 2 concurrent accesses and we can implement LI_0, LI_1 and RI using 2-port memories.

For the FSM state metric, the forward unit writes to the memory while both the backward and LI_0, LI_1 RI update unit read the same data from the memory. As a result, we only need a single bank of 2-port memories.

The channel metric memory is divided into two banks each comprising 1024 entries. The ADC and the acquisition module always work on different banks. By subdividing each bank into two sub-banks, one for the FBA odd segment and the other for the FBA even segment, there are at most two simultaneous accesses to the same segment. Therefore, the channel metrics can also be stored using 2-port memories. In order to reuse the state metric memory once the backward metric is computed, the FSM state metrics are stored in the physical memory in reverse order for even segments. For example, the even segment F_{128} to F_{255} are stored in the state metric memory [127:0] while the odd segment F_0 to F_{127} are stored in the state metric memory [0:127]. The details are also shown in Fig. 11.

For design simplicity, we used 2-port memories in our prototype implementation since it is free in our target device (Xilinx Virtex II FPGA). However, the design can be easily ported to single port memory only architecture by doubling the bus width and time division multiplexing the access.

E. Verification Unit

Our verification unit, shown in Fig. 13, consists of two parts, a PN sequence extrapolation unit and a correlator unit. The extrapolation unit extends the 22-bit PN estimate it receives to the whole observation window. The correlation unit then correlates this sequence with the channel metric. To improve efficiency, the correlator output is checked every $\frac{M}{4}$ pulses and it must exceed the check point threshold before continuing. If the final correlation value exceeds the final threshold, acquisition is declared. The final threshold is chosen to be $0.65 \cdot q \cdot 1024$, which was found by simulation. This yields good acquisition performance as shown in Fig. 12 and the frequency of false alarm is 0 in 5000 trials when the signal is absent.

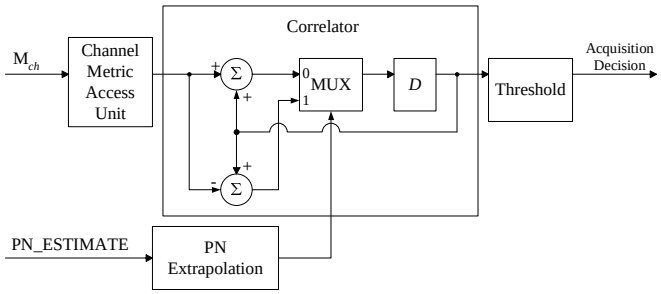


Fig. 13. Verification unit for the PN acquisition module.

F. Hardware Implementation

We implemented the architecture using Verilog HDL. The code is synthesized by Synplicity, then mapped by Xilinx Foundation to a Xilinx Virtex 2 device (XC2v250-6). The number of bits implemented in block RAM is 28160, the number of 4-input LUTs used is 1621 and the number of slices used is 1039. The design can run at 73 MHz. These figures show that memory is the main component of the circuit and justify our decision to trade off logic for memory reduction.

Our baseline design can decode $\frac{Freq_{clk}}{15}$ pulses per second. Assuming a 60 MHz clock, our prototype generates a PN code phase decision every $\frac{15}{60\ MHz} \cdot 1024 = 2.56\ \mu s$. The decode process has to be repeated for each frame epoch estimate until the correct frame epoch is found. Assuming the frame time $T_f = 250$ ns (i.e., pulse rate = 4 Mpulses/s) and pulse width $T_p = 1.6$ ns, the approximate average acquisition time of our prototype system is $T_{acq} = 2.56\ \mu s \cdot \frac{T_f}{T_p} \cdot 0.5 = 20$ ms with $P_{acq} = 0.95$ at $\frac{E_c}{N_0} = -8.9$ dB. This assumes that half of the frame epoch values are searched on average.

As a comparison, to achieve the same average T_{acq}, hardware based on parallel search and running at the same frequency requires approximately 5.6×10^5 correlators, 5.6×10^5 14-bit comparators and 5.6×10^5 4-bit registers. For serial search, the hardware is trivial if we assume a one addition per clock architecture. However, it takes an average of 5.6×10^3 s (approximately 1.5 hours) to acquire the PN sequence if running at the same frequency.

To further lower T_{acq}, we can use parallel FBA architectures (i.e., instantiating multiple forward and backward units to process multiple data segments in parallel). We expect that the increase in logic will be approximately linear when the speed up factor does not exceed 8 because we already divide the observation window into 8 segments in our iterative decoder and each of them can be run in parallel. For lower

speed applications, our design can be further simplified to using single port memory and running the update sequentially. Such a design can save in the number of adders and reduce the routing resources. Therefore we expect the logic gate count will scale linearly for target pulse rate varies from 500 kpulses/s to 32 Mpulses/s.

Our design can also be directly extended to operate at even lower SNR. This requires adding auxiliary model decoders as well as memories for saving the messages from the additional decoders. Since a 6^{th} order model is approximately three times more complex than a 2^{nd} order model, we estimate that the operating E_c/N_0 can be lowered to -13 dB by tripling the gate count or alternatively, increasing the acquisition time by 3 times and tripling the message memory.

V. Conclusion

In this paper, we present a new hardware architecture for fast PN acquisition in UWB systems based on iterative message passing on a graphical model with redundancies. Our new algorithm improves sensitivity significantly via the introduction of multiple redundant models. Hardware based on the algorithm is economical to implement and can rapidly acquire very long PN sequences. There is no known way to accomplish this with traditional approaches using similar hardware resources.

We examined in detail the design trade-offs in choosing an appropriate architecture for the main component: a forward backward algorithm based decoder. We then demonstrated how to combine multiple redundant models into a single model to reduce memory usage. Finally, we gave a detailed account on our hardware implementation and discuss various implementation techniques. Our design can be fit to a small FPGA while full parallel search is impractical to implement and serial search is fiver orders of magnitude slower than our design.

Future work will be focused on designing hardware for a more realistic system model which incorporates oversampling, interference and multi-path channel distortions.

Appendix
Update Equations for the 4-State FSM Decoder

The update equations for the FSM are direct application of the standard message passing algorithms [21] on Fig. 10(c). The variables F_k, B_k, $\mathrm{LI\_0}_k$, $\mathrm{LI\_1}_k$ and RI_k are defined in Fig. 10(d). Alternatively, they can be derived from Fig. 10(b) by applying standard SISO update rules to each SISO. We list the equations based on Fig. 10(c) because it is easier to compare against [2, (23)] - [2, (29)].

$$\mathrm{F}_0 = 0 \tag{14}$$

$$\mathrm{B}_M = 0 \tag{15}$$

$$\mathrm{F}_{k+1}[0] = \min(\mathrm{F}_k[0],\ \mathrm{F}_k[2] + \mathrm{LI\_1}_k) \tag{16}$$

$$\begin{aligned}\mathrm{F}_{k+1}[1] = \min(&\mathrm{F}_k[0] + \mathrm{RI}_k + \mathrm{LI\_0}_k + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[2] + \mathrm{RI}_k + \mathrm{LI\_0}_k)\end{aligned} \tag{17}$$

$$\mathrm{F}_{k+1}[2] = \min(\mathrm{F}_k[1] + \mathrm{LI\_0}_k,\ \mathrm{F}_k[3] + \mathrm{LI\_0}_k + \mathrm{LI\_1}_k) \tag{18}$$

$$\mathrm{F}_{k+1}[3] = \min(\mathrm{F}_k[1] + \mathrm{RI}_k + \mathrm{LI\_1}_k,\ \mathrm{F}_k[3] + \mathrm{RI}_k) \tag{19}$$

$$\mathrm{B}_{k-1}[0] = \min(\mathrm{B}_k[0],\ \mathrm{B}_k[1] + \mathrm{RI}_k + \mathrm{LI\_0}_k + \mathrm{LI\_1}_k) \tag{20}$$

$$\mathrm{B}_{k-1}[1] = \min(\mathrm{B}_k[2] + \mathrm{LI\_0}_k,\ \mathrm{B}_k[3] + \mathrm{RI}_k + \mathrm{LI\_1}_k) \tag{21}$$

$$\mathrm{B}_{k-1}[2] = \min(\mathrm{B}_k[0] + \mathrm{LI\_1}_k,\ \mathrm{B}_k[1] + \mathrm{RI}_k + \mathrm{LI\_0}_k) \tag{22}$$

$$\mathrm{B}_{k-1}[3] = \min(\mathrm{B}_k[2] + \mathrm{LI\_0}_k + \mathrm{LI\_1}_k,\ \mathrm{B}_k[3] + \mathrm{RI}_k) \tag{23}$$

$$\begin{aligned}\mathrm{LO\_0}_k = \min(&\mathrm{F}_k[0] + \mathrm{B}_{k+1}[1] + \mathrm{RI}_k + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[1] + \mathrm{B}_{k+1}[2],\\ &\mathrm{F}_k[2] + \mathrm{B}_{k+1}[1] + \mathrm{RI}_k,\\ &\mathrm{F}_k[3] + \mathrm{B}_{k+1}[2] + \mathrm{LI\_1}_k)\\ -\min(&\mathrm{F}_k[0] + \mathrm{B}_{k+1}[0],\\ &\mathrm{F}_k[1] + \mathrm{B}_{k+1}[3] + \mathrm{RI}_k + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[2] + \mathrm{B}_{k+1}[0] + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[3] + \mathrm{B}_{k+1}[3] + \mathrm{RI}_k)\end{aligned} \tag{24}$$

$$\begin{aligned}\mathrm{LO\_1}_k = \min(&\mathrm{F}_k[0] + \mathrm{B}_{k+1}[1] + \mathrm{RI}_k + \mathrm{LI\_0}_k,\\ &\mathrm{F}_k[1] + \mathrm{B}_{k+1}[3] + \mathrm{RI}_k,\\ &\mathrm{F}_k[2] + \mathrm{B}_{k+1}[0],\\ &\mathrm{F}_k[3] + \mathrm{B}_{k+1}[2] + \mathrm{LI\_0}_k)\\ -\min(&\mathrm{F}_k[0] + \mathrm{B}_{k+1}[0],\\ &\mathrm{F}_k[1] + \mathrm{B}_{k+1}[2] + \mathrm{LI\_0}_k,\\ &\mathrm{F}_k[2] + \mathrm{B}_{k+1}[1] + \mathrm{RI}_k + \mathrm{LI\_0}_k,\\ &\mathrm{F}_k[3] + \mathrm{B}_{k+1}[3] + \mathrm{RI}_k)\end{aligned} \tag{25}$$

$$\begin{aligned}\mathrm{RO}_k = \min(&\mathrm{F}_k[0] + \mathrm{B}_{k+1}[1] + \mathrm{LI\_0}_k + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[1] + \mathrm{B}_{k+1}[3] + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[2] + \mathrm{B}_{k+1}[1] + \mathrm{LI\_0}_k,\\ &\mathrm{F}_k[3] + \mathrm{B}_{k+1}[3])\\ -\min(&\mathrm{F}_k[0] + \mathrm{B}_{k+1}[0],\\ &\mathrm{F}_k[1] + \mathrm{B}_{k+1}[2] + \mathrm{LI\_0}_k,\\ &\mathrm{F}_k[2] + \mathrm{B}_{k+1}[0] + \mathrm{LI\_1}_k,\\ &\mathrm{F}_k[3] + \mathrm{B}_{k+1}[2] + \mathrm{LI\_0}_k + \mathrm{LI\_1}_k)\end{aligned} \tag{26}$$

$$\mathrm{RI}_k = \mathrm{LO\_0}_{k+22} + \mathrm{LO\_1}_{k+44} + \mathrm{M}_{ch}[k] \tag{27}$$

$$\mathrm{LI\_0}_{k+22} = \mathrm{RO}_k + \mathrm{LO\_1}_{k+44} + \mathrm{M}_{ch}[k] \tag{28}$$

$$\mathrm{LI\_1}_{k+44} = \mathrm{RO}_k + \mathrm{LO\_0}_{k+22} + \mathrm{M}_{ch}[k] \tag{29}$$

$$\mathrm{M}_{dec} = \mathrm{RO}_k + \mathrm{LO\_0}_{k+22} + \mathrm{LO\_1}_{k+44} + \mathrm{M}_{ch}[k] \tag{30}$$

Acknowledgement

The authors would like to thank Mingrui Zhu for helpful discussion on this research and providing the simulation results for the baseline iMPA algorithm in [2] shown in Fig. 5.

References

[1] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt, *Spread Spectrum Communications Handbook*. McGraw-Hill, 1994.

[2] K. M. Chugg and M. Zhu, "A new approach to rapid PN code acquisition using interative message passing techniques," *IEEE J. Select. Areas Commun.*, vol. 23, no. 51, June 2005.

[3] E. A. Homier and R. A. Scholtz, "Rapid acquisition of ultra-wideband signals in the dense multipath channel," in *Digest of Papers, IEEE Conference on Ultra Wideband Systems and Technologies*, May 2002, pp. 105 – 109.

[4] M. Zhu and K. M. Chugg, "Iterative message passing techniques for rapid PN code acquisition," in *Proc. IEEE Military Comm. Conf.*, Boston, MA, October 2003, pp. 434–439.
[5] I. D. O'Donnell, S. W. Chen, B. T. Wang, and R. W. Brodersen, "An integrated, low power, ultra-wideband reansceriver architecture for low-rate, indoor wireless systems," *IEEE CAS Workshop on Wireless Communications and Networking*, September 2002.
[6] A. Polydoros and C. L. Weber, "A unified approach to serial search spread-spectrum code acquisition," *IEEE Trans. Commununication*, vol. vol. 32, pp. 542–560, May 1984.
[7] L. Yang and L. Hanzo, "Iterative soft sequential estimation assisted acquisition of m-sequence," *IEE Electronics Letters*, vol. 38, no. 24, pp. 1550 – 1551, Nov 2002.
[8] ——, "Acquisition of m-Sequences using recursive soft sequential estimation," *IEEE Trans. Commununication*, vol. 52, no. 2, pp. 199 – 204, Feb 2004.
[9] B. Vigoda, J. Dauwels, N. Gershenfeld, and H. Loeliger, "Low-complexity LFSR synchronization by forward-only message passing," *IEEE Trans. Information Theory*, June 2003, submitted.
[10] S. W. Golomb, *Shift Register Sequences, revised edition*. Aegean Park Press, 1982.
[11] C. Berrou, A. Glavieux, and P. Thitmajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *Proc. International Conf. Communications*, Geneva, Switzerland, May 1993, pp. 1064–1070.
[12] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commununication*, vol. 44, no. 10, pp. 1261–1271, October 1996.
[13] R. G. Gallager, "Low density parity check codes," *IEEE Trans. Information Theory*, vol. 8, pp. 21–28, January 1962.
[14] ——, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
[15] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEE Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, August 1996.
[16] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Linköping University (Sweden), 1996.
[17] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 140–152, February 1998.
[18] S. M. Aji, "Graphical models and iterative decoding," Ph.D. dissertation, California Institute of Technology, 1999.
[19] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Information Theory*, vol. 46, no. 2, pp. 325–343, March 2000.
[20] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Information Theory*, vol. 47, pp. 498–519, Feb. 2001.
[21] K. M. Chugg, A. Anastasopoulos, and X. Chen, *Iterative Detection: Adaptivity, Complexity Reduction, and Applications*. Kluwer Academic Publishers, 2001.
[22] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commununication*, vol. 44, pp. 591–600, May 1996.
[23] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. Information Theory*, vol. 44, no. 3, pp. 909–926, May 1998.
[24] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Information Theory*, vol. IT-27, pp. 533–547, September 1981.
[25] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-input soft-output modules for the construction and distributed iterative decoding of code networks," *European Trans. Telecommun.*, vol. 9, no. 2, pp. 155–172, March/April 1998.
[26] J. S. Yedidia, J. Chen, and M. Fossorier, "Generating code representations suitable for belief propagation decoding," in *Proc. 40-th Allerton Conference Commun., Control, and Computing*, Monticello, IL., October 2002.
[27] N. Santhi and A. Vardy, "On the effect of parity-check weights in iterative decoding," in *Proc. IEEE Internat. Symp. Information Theory*, Chicago, IL., July 2004, p. 322.
[28] M. Schwartz and A. Vardy, "On the stopping distance and the stopping redundancy of codes," *IEEE Trans. Information Theory*, 2005, submitted.
[29] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory*, vol. IT-20, pp. 284–287, March 1974.
[30] A. J. Viterbi, "Justification and implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, February 1998.
[31] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Trans. VLSI*, vol. 7, no. 3, September 1999.
[32] J. Dielissen and J. Huisken, "State vector reduction for initialization of sliding windows MAP," in *2nd Internation Symposium on Turbo Codes & Related Topics*, Brest, France, 2000, pp. 387 – 390.
[33] S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Communications Letters*, vol. 6, no. 7, pp. 288 – 290, July 2002.
[34] A. Abbasfar and K. Yao, "An efficient and practical architecture for high speed turbo decoders," in *IEEE 58th Vehicular Technology Conference*, October 2003, pp. 337 – 341 Vol.1.
[35] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. Commununication*, vol. 37, no. 11, November 1989.