VLSI ARCHITECTURE OF A WIRELESS CHANNEL ESTIMATOR USING SEQUENTIAL MONTE CARLO METHODS

Mahdi Shabany, Hassan Shojania, Jing Zhang, Javad Omidi, P. Glenn Gulak

Electrical and Computer Engineering Department
University of Toronto

ABSTRACT

The regular and repetitive nature of the Sequential Monte Carlo (SMC) method makes it very attractive for implementation using parallel and pipelined architectures. This paper develops a VLSI architecture for the hardware implementation of the SMC algorithm using bootstrap filter. A flat fading wireless channel is considered as our framework on which the channel estimator is designed and implemented using the SMC method. The design and verification activities at the algorithm level, architecture level, and the circuit level are reviewed. The proposed architecture is verified with an FPGA implementation.

1. INTRODUCTION

A state space model is used in many mobile communication systems to analyze the dynamic behavior of the system and to provide tools to estimate and track variations in the system parameters. Dynamic system modeling described by a state space model has attracted much attention from researchers in different fields. In the Bayesian approach to dynamic state estimation, one attempts to construct the *posterior* probability density function (pdf) of the state based on all available information, including the set of received measurements. Since this pdf embodies all available statistical information, it may be said to be the complete solution to the estimation problem.

In the classic state space model estimation using a Kalman filter, the posterior probability density over the parameters to be estimated is assumed to be Gaussian. The mean and covariance matrix are propagated using recursive update equations each time new data is received. However, the derivation of the Kalman model is based on the linearity of the model and the Gaussianity of both the dynamic noise in the process equation and the measurement noise in the measurement equation [1]. Unfortunately, many of the state estimation problems in practice are nonlinear and non-Gaussian which limits the practical usefulness of the classical Kalman filter (see [2] for a survey). The Extended Kalman Filter (EKF) method can be applied as an approximation for nonlinear systems based on the local linearization of the system. However, the EKF always approximates the posterior density to be a Gaussian. Therefore, if the true density is non-Gaussian, an EKF would not be a good approach.

Due to the intrinsic limitations in the above mentioned methods, sequential state estimation, which overcomes these limitations, becomes central to the study of general dynamical systems. Specifically, Sequential Monte Carlo (SMC) estimation is a technique to implement a recursive Bayesian filter by Monte Carlo simulations. The key idea is to represent the required posterior density function by a set of random samples with associated weights and to compute estimates based on these samples and weights. Thereby, it provides a convenient approach for dealing with multimodal distributions, nonlinear dynamics, and observation equations. In fact, SMC is a promising advanced solution for various estimation problems.

The regular and repetitive nature of the SMC method makes it very attractive for implementation using parallel and pipelined VLSI architectures. The primary purpose of this paper is to propose a VLSI architecture for the implementation of the SMC algorithm. A flat fading wireless channel is considered and a channel estimator is designed and implemented. Hereafter, we refer to our design as a Sequential Monte Carlo Channel Estimator (SMCCE).

2. THE SMC FRAMEWORK

The system equation and the measurement equation are two constituents of a dynamic system model which are shown in (1) and (2), respectively.

$$h(t) = f_t\big[h(t-1:t-m), \psi(t)\big] \tag{1}$$

$$y(t) = g_t\big[h(t), v(t)\big] \tag{2}$$

where $h(t:t-m) = \{h(t), h(t-1), ..., h(t-m)\}$ is the state sequence, $\psi(t)$ is the process noise, $y(t)$ is the observed data, and $v(t)$ is the measurement noise.

This work was supported by Natural Sciences and Engineering Research Council of Canada (NSERC).

Table 1. The SMCCE Algorithm.

Step I. Draw estimation vectors $\{h^i(t)\}_{i=1}^N$ for initial time steps $t = \{1, 2, 3\}$.

For each time step $t = \{4, 5, ...\}$:
Step II. Pick a set of process noise samples $\{\psi^i(t)\}_{i=1}^N$ (random numbers) from a Gaussian distribution with variance Q.

Step III. Calculate estimation vector, $\hat{H} = \{\hat{h}^i(t)\}_{i=1}^N$, using:

$$\hat{h}^i(t) = -Ah^i(t-1) - Bh^i(t-2) - Ch^i(t-3) + D\psi^i(t).$$

Step IV. Calculate the vector of estimated received signal: $Y = \{y^i(t)\}_{i=1}^N$ as $y^i(t) = d(t) \cdot \hat{h}^i(t)$.

Step V*. Calculate posteriori distribution of the estimation vector using: $p_i(t) = e^{\frac{-1}{2R}(y_{rcv}(t) - y^i(t))^2}$.

Step VI. Calculate the *importance weight* vector $\{\omega_t^i\}_{i=1}^N$ using: $\omega_t^i = \lfloor \frac{Np_j(t)}{\sum_{i=1}^N p_i(t)} \rfloor$.

Step VII. Update the state estimation vector $\hat{H}$:
The *importance weights* vector, $\{\omega_t^i\}_{i=1}^N$, is used to filter/resample the elements of $\hat{H}$ to construct the updated vector $H = \{h^i(t)\}_{i=1}^N$.

Step VIII. The final value $h(t)$ is calculated by (6).

End for

* R is the variance of the measurement noise.

In the SMC framework, the posterior density function is approximated by a set of random samples with associated weights [2]. With reference to the state space equations in (1) and (2), the posterior pdf of interest is $p\big[h(t)|y(t:1)\big]$ which is used to estimate $h(t)$. Let $\{h^i(t:0), \omega_t^i\}$ denote a set of random samples and their associated *importance weights* drawn with respect to this posterior pdf. Therefore, the discrete weighted approximation of $p(h(t)|y(t:1))$ is:

$$p\big[h(t)|y(t:1)\big] = \sum_{i=1}^{N} \omega_t^i \delta\big[h(t:0) - h^i(t:0)\big] \qquad (3)$$

The challenge is how we can draw proper samples from the posterior pdf and determine their associated weights. The *bootstrap* filter is proposed in [3] as a simple method for online estimation in a state space model. It is shown that if $\{h^i(t-1:0), \omega_{t-1}^i\}_{i=1}^N$ is the sample-weight set following the posterior distribution $p\big[h(t-1)|y(t-1:1)\big]$ and N is large enough, then using the updating procedure the sample-importance weight vector $\{h^i(t:0), \omega_t^i\}$ would follow the posterior pdf $p\big[h(t)|y(t:1)\big]$. The procedure at time t is as follows:

1. Draw $\hat{h}^i(t)$ from the state equation $f_t\big[h(t)|h^i(t-1: t-m)\big], i = 1, ..., N$.
2. Weight each draw by $\omega_t^i \propto g_t\big[y(t)|\hat{h}^i(t)\big]$.
3. Resample from $\{\hat{h}^i(t)\}_{i=1}^N$ with probability proportional to ω_t^i to produce a random sample $\{h^i(t)\}_{i=1}^N$.

The purpose of resampling is to reduce the variance of samples which increases over time.

3. SYSTEM MODEL AND ALGORITHM ANALYSIS

In wireless communication systems, the transmitted signal is corrupted by the fading characteristic of the channel along with the received additive noise. To correctly recover the information in the transmitted signal by observing the received signal, it is important to know the Channel Impulse Response (CIR) which is known by channel estimation.

The fading channel can be modeled as a linear time-varying system. In fact, the fading characteristic of the channel can be modeled by a Gaussian noise process, namely the process noise $\psi(t)$ with a known variance of Q, fed into a typical low-pass filter such as a third-order Butterworth filter. Therefore, the frequency response and the impulse response of the channel can be described by an auto-regressive moving-average (ARMA) model as:

$$h(t) = -Ah(t-1) - Bh(t-2) - Ch(t-3) + D\psi(t) \quad (4)$$

where the coefficients A, B, C, and D are determined based on the maximum Doppler frequency shift in the system along with the sampling frequency of the channel. Their values based on the assumption of the maximum fading rate of $f_dT = 0.05$ are -2.8174, $+2.6593$, -0.8398 and $+0.002$, respectively. Using this channel, the received signal, $y(t)$, can be expressed as:

$$y(t) = d(t) * h(t) + v(t) \qquad (5)$$

Note that equations (4) and (5) consist of the state space model in our system corresponding to the general model of (1) and (2), respectively. Assuming a known transmitted signal, such as a *pilot* signal, channel estimation is performed based on the observed signal $y(t)$. Using the updating procedure steps described in section 2, Table. 1 shows the computation flow of this algorithm.

Meanwhile, the final value of the channel estimation for time t is calculated as the weighted average of N estimated samples as:

$$h(t) = \frac{\sum_{i=1}^N \omega_t^i h^i(t)}{\sum_{i=1}^N \omega_t^i} \qquad (6)$$

Note that $\sum_{i=1}^N \omega_t^i \leq N$. Therefore, to have exactly N samples we simply repeat the last element.

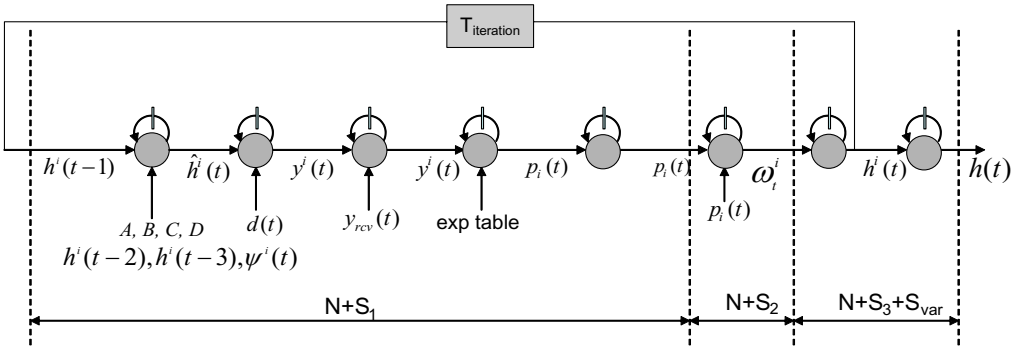

Fig. 1. Architecture I.

4. ARCHITECTURE EXPLORATION

We consider two architectures proposed in this paper. We first address Architecture *I* in order to appreciate the principal concepts in the architecture realization as well as sequential dependencies intrinsic to the algorithm. Then Architecture *II* is described which is designed to solve the weaknesses of the architecture *I* yielding a more efficient realization.

4.1. Architecture I:

As shown in Fig. 1, Architecture *I* consists of three cascaded pipeline *sections*. Each pipeline section consists of internal pipeline *stages*. The proposed architecture performs the SMCCE algorithm proposed in Fig. 1 as follows. In the pipeline section one, using the previous estimated H tables, the estimated CIR, $\left\{\hat{h}^i(t)\right\}_{i=1}^{N}$, at time t is calculated (step III). Then the estimated samples, $\left\{y^i(t)\right\}_{i=1}^{N}$, are determined (step IV) and their corresponding *a posterior* distribution, $p_i(t)$s is calculated (step V). Finally, their summation is determined. In the second pipeline section, the importance weights, $\left\{\omega_t^i\right\}_{i=1}^{N}$, are calculated (step VI). Eventually, the third pipeline section updates the state estimation vector (step VII) and then calculates the final value for CIR, $h(t)$, (step VIII). Note that to calculate $p_i(t)$ we need to implement the exponential function. However, we use a lookup table in order to reduce the complexity of the design while considering the performance and precision.

In fact, the relationship between pipeline sections 2 and 3 is "producer-consumer". Thus, we can use a lookup table to pass the weights between sections 2 and 3 (i.e., start 3 only after 2 is finished). The overall processing time for an iteration period, i.e. the time required to generate the final estimated $h(t)$, is

$$T_{iteration} = (N+S_1+N+S_2+N+S_3+S_{var})*T_p \quad (7)$$

where N is the sample space (i.e. 500 in our example), S_1, S_2, S_3 are the time latency in terms of number of clock cycles for three consecutive pipeline sections, respectively, T_p is the clock period, and S_{var} is the variable processing time of pipeline section 3. To understand why S_{var} should be taken into account, let's consider an extreme case where all ω_t^i are zero except the last one, (i.e. $\omega_t^N \neq 0$). In this situation it takes $N-1+S_3$ clock cycles to start processing the last sample. Since $\omega_t^N = N$, the N^{th} item will be repeated N times. Therefore, it takes N extra cycles to write these samples into the memory block. This worst case scenario yields:

$$S_{var} \leq N \Rightarrow \max\{T_{iteration}\} = 4N + S_1 + S_2 + S_3 \quad (8)$$

According to Fig. 1, this architecture is simple to implement. The problem is that the calculation time for pipeline section 3 is variable. Moreover, this architecture does not make the best possible use of intrinsic parallelism (i.e. due to the usage of a lookup table between pipeline sections 2 and 3, the start of the calculation in pipeline section 3 has been delayed until all entries of the table are available). Conceptually, referring to Fig. 1, the minimum required time steps to do the whole calculation to determine $h(t)$ is $T_{iteration} \geq N + S_1 + N + S_2 + S_3$[1]. Therefore, another architecture can be proposed to eliminate the extra terms to save time and eventually lead to a faster system.

4.2. Architecture II:

In Architecture *II*, the design makes use of a pipelined elastic buffer, eliminating N extra cycles. This means that once the pipeline section 3 has enough information, it can start to work. Using this idea , there is a "time saving" interconnection between two last pipeline sections. Moreover, this approach leads to a "memory saving" architecture as well, because there is no need to save ω_t^i anymore.

[1] $S_1 + S_2 + S_3$ is the number of clock cycles related to time latency that is inevitable, $2N$ is the time required to calculate all $p_i(t)$ (pipeline section one) and all ω_t^i (pipeline section three).

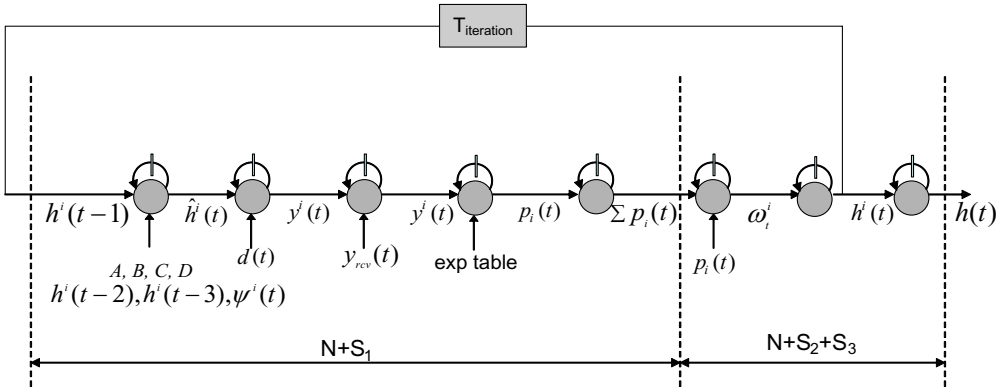

Fig. 2. Architecture II.

Secondly, to improve the performance, S_{var} should be dropped because if the entries are written in the memory one by one then the calculation time for pipeline section 3 would be variable as pointed by (8). This variable time would force us to account for the worst case delay (according to (8)) limiting potential maximum throughput. Instead, we propose a coded architecture to improve the data structure of $\left\{h^i(t)\right\}_{i=1}^{N}$ such that for multiple entries with the same value of $h^i(t)$, only one write to the memory is required. To do so, one bit, called the pointer bit, is inserted at the beginning of each record of $h^i(t)$. When the pointer bit is "1", that record is considered as a new record and will be used, otherwise the record will be ignored and the last valid record will be used instead. With respect to the worst case example mentioned in Architecture *I*, when $\omega_t^i = 0$ for all i except $i = N$, nothing is written into the memory until the last record is read. Therefore, the last record corresponding to $i = N$ is written into the first row of the memory. Thus, one write is required and $S_{var} = 0$.

To make this coded architecture work, in pipeline section one, after each read from H tables, using the "write after read" capability of the dual port memories, the pointer bits of all entries of $\left\{h^i(t-3)\right\}_{i=1}^{N}$ table are set to zero so they are ready to be overwritten in the next time step. Since architecture *II* is the cascade of two pipeline sections, the overall iteration period is $T_{iteration} = (2N + S_1 + S_2 + S_3) * T_p$.

This architecture is simple and has a fixed calculation time. Further improvements can be made by employing multiple parallel pipelines to increase system throughput. Note this would introduce its own complexities; e.g. the need to break $\left\{h^i(t)\right\}_{i=1}^{N}$ tables into multiple tables (each processed by one of the parallel pipelines). This is because Block RAMs (BRAM) in typical FPGAs are dual ported and they don't allow more than two concurrent accesses. Here, without loss of generality, we consider a single pipeline stream.

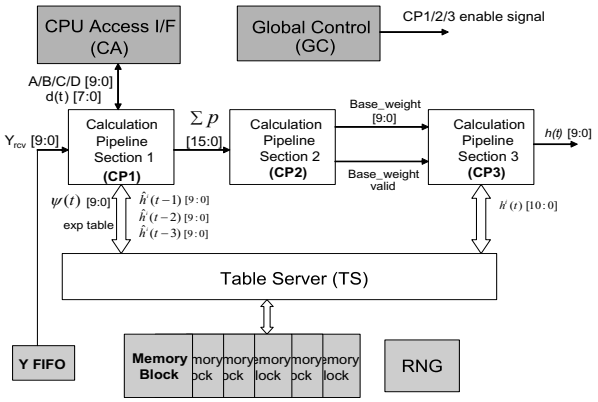

Fig. 3. Physical block diagram of the system.

5. ARCHITECTURE SPECIFICATION

The implementation of the SMCCE consists of two major parts, i.e. the Calculation Core and the CPU Bus Interface as shown in Fig. 3. The Calculation Core is further divided into the calculation pipelines (CP1, CP2, CP3), Table Server (TS), Global Control logic (GC), and instantiated block memory, which are shown in Fig. 3. The CPU Interface provides an interface between the On-Chip Peripheral Bus (OPB-Bus) and internal accessible registers and memories. In Fig. 3, the Random Number Generator (RNG) generates random numbers with Gaussian distribution, and the table server coordinates all table accesses. Fig. 3 also shows the word length of selected variables in the design. These values are selected to minimize the memory requirements while maintaining the good performance.

6. DESIGN CHARACTERISTICS

Verification of the calculation core was done through a set of testbenches written in Verilog while the complete core was verified with IBM's OPB Bus Functional Model Toolkit. Xilinx Virtex-II Multimedia Board with XC2V2000 FF896

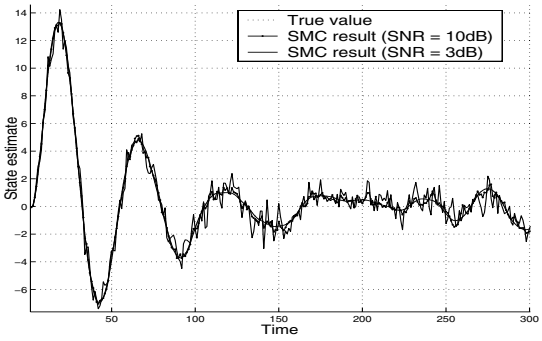

Fig. 4. The channel estimation results for SNR = 3dB and 10dB.

FPGA was our implementation platform. Since enough on-chip Block RAM was available, external memory was not used. The whole embedded system occupies around 21 percent of XC2V2000 including the soft processor (2264 out of 10752 available slices where each slice includes two Look-Up Tables (LUT)). Out of the available 56 blocks of 18 KBits BRAMs, 7 were used by the SMCCE core to hold different tables and the rest were available for the MicroBlaze system software. A MicroBlaze soft-processor was used in the final system verification to stream data into and out of the SMCCE core as fast as possible.

Using Architecture II, each iteration takes $2N + S_1 + S_2 + S_3 = 2N + S_{pipeline-overhead}$ cycles. The total overhead is 39 cycles so each iteration takes 1039 cycles for $N = 500$. Since the calculation core is running at 135 MHz (5 times OPB bus clock of 27 MHz), ideally a throughput of 129.93 K iterations/second can be reached. Software overhead shouldn't prevent us from reaching this rate as the input/output FIFO mechanism and interrupt-based notification to the CPU should minimize the overhead. Higher estimation rates (for other applications) can be enabled with a custom hardware solution.

7. SIMULATION RESULTS

To verify the performance of our design, the channel model introduced in section 3 was employed to examine the estimation results. The linear Gaussian environment was considered to simplify the proof of concept. A random process noise with variance $Q = 15$ was chosen to model the channel characteristics. The objective is to estimate a series of consecutive samples in time. The first 300 samples are as in Fig. 4 where the true value as well as the estimated results for SNR 3dB and 10dB are shown. Using the weighted average based on (6), the statistical comparison of the estimated channel states with the true values is shown in Fig. 5a. To further elucidate the estimation accuracy, the posterior densities used for estimation in each time instant are shown in Fig. 5b for the first 100 samples. It is worth noting the close

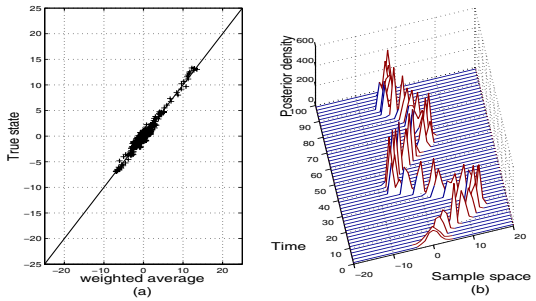

Fig. 5. The weighted average results and the final posterior distributions.

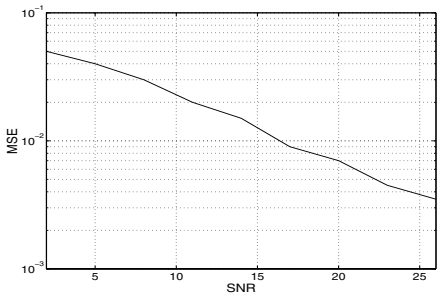

Fig. 6. Mean Square Error for various SNR values.

correspondence between these posteriors and the results in Fig. 4. The Mean Square Error (MSE) of the implemented architecture for various SNR values is also shown in Fig. 6.

8. CONCLUSIONS

In this paper, a VLSI architecture for the implementation of the SMC algorithm using a Bayesian bootstrap filter framework was proposed. A flat fading wireless channel was considered as our framework and a channel estimator was designed and implemented based on the SMC method to track the CIR. The paper covers the design and verification activities from the algorithm evaluation through the FPGA implementation. An important step forward would be finding efficient architectures for a more general case.

9. REFERENCES

[1] Y. C. Ho and R. C. K. Lee, "A Bayesian approach to problems in stochastic estimation and control," *IEEE Trans. Automat. Contr.*, vol. AC-9, pp. 333–339, 1964.

[2] A. Doucet, J. F. G. de Freitas, and N. J. Gordon, *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, New York, 2001.

[3] N. Gordon, D. Salmond, and A. F. M. Smith, "Novel approach to non-linear and non-Gaussian Bayesian state estimation," *Proc. Inst. Elect. Eng.*, vol. 140, pp. 107–113, 1993.