A Complete Real-Time 802.11a Baseband Receiver Implemented on an Array of Programmable Processors

Anh T. Tran, Dean N. Truong, and Bevan M. Baas
University of California - Davis
{*anhtr, hottruong, bbaas*}*@ucdavis.edu*

***Abstract*—This paper reports the design and software implementation of a real-time digital baseband receiver compliant with the IEEE 802.11a standard on the AsAP2 platform, a DSP chip multiprocessor. The computational platform consists of an array of programmable processors and configurable accelerators interconnected in a 2-D mesh network that are well matched for implementing complex DSP and embedded systems such as wireless and video applications. The receiver has full functionality including frame detection, timing synchronization, carrier frequency offset compensation, and channel equalization. It supports all eight operational modes defined in the standard: 6, 9, 12, 18, 24, 36, 48 and 54 Mbps. The implementation is optimized such that the receiver can obtain a full 54 Mbps rate while using an array of 29 small processors plus Viterbi and FFT accelerators configured to operate at 590 MHz.**

***Index Terms*—WLAN, IEEE 802.11a, baseband receiver, SDR, DSP, multiprocessing.**

I. Introduction

In recent years, a wide variety of wireless communication systems has come into widespread use. To quickly adapt with this variety the flexibility of software-defined radio (SDR) solutions have become increasingly attractive. Some SDR platforms were proposed and the 802.11a system [1] is one of the wireless protocols used as benchmarks to evaluate their efficiency.

Some real-time software implementations of the 802.11a baseband transmitter were reported in the literature [2]–[4]. The implementations of its receiver, which is much more complicated, however, either cannot obtain a 54 Mbps throughput [4]–[7], or ignore some necessary features such as frame detection/synchronization, carrier frequency offset estimation/compensation and channel equalization [8]–[10]. Speeding up the performance can be obtained by using some built-in dedicated hardware such as Viterbi decoder and FFT for the core computational components. However, other components in the system which are implemented in software become bottlenecks that limit the throughput. We eliminate these bottlenecks to get a complete real-time receiver by exploiting the advantage of task-level parallelism on multiple small cores of the AsAP2 platform [11], a second generation of the **As**ynchronous **A**rray of Simple **P**rocessors [12].

The AsAP2 many-core platform includes an array of 164 simple fine-grain single-issue 6-stage processors fabricated in 65 nm CMOS. Each processor has its own 128x35-bit instruction memory, 128x16-bit data memory and two 64x16-bit FIFOs. These processors only support the basic 16-bit arithmetic and logic instructions. Complex arithmetic operations such as CORDIC rotation, trigonometric and square root functions are implemented by a sequence of many basic instructions. The latency of these operations can be significantly reduced by making them execute in parallel on as many processors as possible. The simplicity keeps the processor's area small that is only 0.17 mm^2 each.

The Viterbi and FFT accelerators were also added into the platform to speed up the core computational components in wireless applications. The Viterbi can decode convolution codes up to a constraint length of 10 and the FFT is capable of performing FFT/IFFT transforms up to 4096-point. These accelerators can be configured in run-time depending on the requirements of the applications. All processors and accelerators are asynchronously interconnected in a statically circuit-switched 2-D mesh network that allows them to operate in different clock domains with frequencies up to 1.2 GHz. The network also supports direct long-distance communication (further than nearest-neighbor connections) that simplifies the mapping of complex applications.

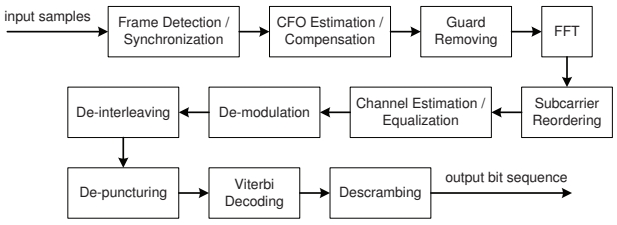

Fig. 1. Block diagram of a complete 802.11a baseband receiver.

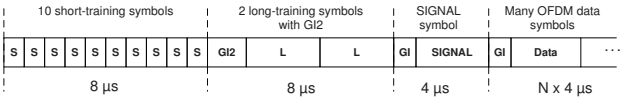

Fig. 2. Structure of a received frame. S: 16-sample short-training symbol; GI2: 32-sample double guard interval; L: 64-sample long-training symbol; GI: 16-sample single guard interval; SIGNAL and DATA fields: 64 samples each.

The outline of this paper is organized as follows. Section II describes the design of a complete 802.11a baseband receiver. The mapping of this receiver on the AsAP2 platform is shown in Section III. Section IV describes the evaluation and improvement of the receiver's throughput. Finally, paper conclusion and our future work are discussed in Section V.

II. Architecture of a Complete 802.11a Baseband Receiver

Fig. 1 shows the architecture of a complete 802.11a baseband receiver. The baseband receiver gets signal samples from an analog-to-digial converter (ADC) with a sampling frequency of 20 MHz. These signal samples form a frame including training symbols and many OFDM symbols as described in Fig. 2.

Ten periodic short-training symbols in the beginning of frame are used for frame detection and timing synchronization. A common timing metric, which is used for both frame detection and timing synchronization, was proposed by Schmidl and Cox [13]:

$$M(n) = \frac{|P(n)|^2}{Q(n)^2} \quad (1)$$

Where $P(n)$ is the auto-correlation between the received frame and a copy delayed 16 samples from itself. $Q(n)$ is the energy of 16 consecutive samples beginning from the n^{th} sample.

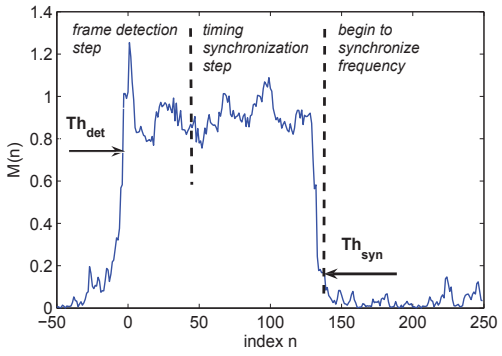

Fig. 3. Plot of the timing metric $M(n)$ with $SNR = 20dB$. Th_{det} and Th_{syn} are thresholds used for frame detection and timing synchronization, respectively.

$$P(n) = \sum_{k=0}^{15} r(n+k+16) \cdot r^*(n+k) \tag{2}$$

$$Q(n) = \sum_{k=0}^{15} |r(n+k)|^2 \tag{3}$$

With $r(n)$ is the received samples and $(.)^*$ denotes the complex conjugate operation.

Since the ten short-training (S) symbols are periodic, under the impact of noise, the timing metric forms an amplitude plateau as described in Fig. 3. This metric begins to increase from the first S symbol, which then fluctuates around a high amplitude level and then gradually decreases at the ninth S symbol. This behavior of the timing metrics allows us to easily set constant threshold values for frame detection and synchronization.

A frame should be detected if its timing metric is larger than a threshold Th_{det} at some consecutive samples (we choose the number of these samples to be 48, which equals the number of samples of 3 S symbols). After the frame is detected, the timing is synchronized by locate the first sample of the timing metric $M(n)$ which is less than the threshold Th_{syn}. This sample is the first sample of the tenth S symbol which allows us to determine all long-training, SIGNAL and OFDM symbols in the frame from now on.

Because division is slow, we replace the frame detection's condition $M(n) = \frac{|P(n)|^2}{Q(n)^2} > Th_{det}$ and the timing synchronization's condition $M(n) = \frac{|P(n)|^2}{Q(n)^2} < Th_{syn}$ with:

$$|P(n)|^2 > Th_{det} \cdot Q(n)^2 \tag{4}$$

and

$$|P(n)|^2 < Th_{syn} \cdot Q(n)^2 \tag{5}$$

respectively. In this project, we choose $Th_{det} = 0.75$ and $Th_{syn} = 0.15$ as results derived by Jiménez et al. [14] and Tang et al. [15].

Once the timing has been synchronized, the frequency synchronization step begins. Due to the difference in the frequency between the transmitter and the receiver, there exists a carrier frequency offset (CFO) between them. This CFO creates a phase error which is accumulated on every sample of the frame. Even when the CFO is small, the phase error still make subcarriers (in the frequency domain) rotate far from the standard constellation points as illustrated in Fig. 4. This CFO extremely degrades the accuracy of the receiver which is why the frequency synchronization step is necessary in a practical receiver.

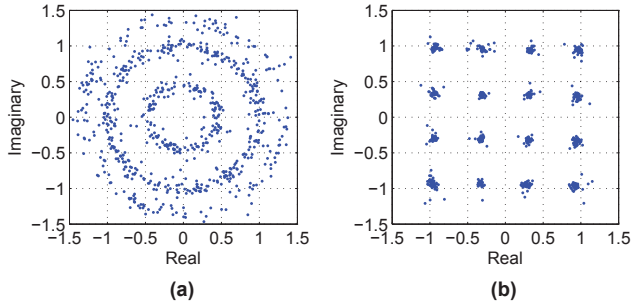

Fig. 4. The constellation of 16-QAM subcarriers in the frequency domain with ϵ = 10 ppm at 5 GHz. a) Without CFO compensation. b) With CFO compensation.

The CFO can be estimated and compensated using two long-training symbols as proposed by Sourour [16]. The phase error caused by CFO is estimated as follows:

$$\alpha = \frac{1}{64} \cdot \angle\{\sum_{k=0}^{63} L_2(k) \cdot L_1^*(k)\} \tag{6}$$

where $L_1(k)$ and $L_2(k)$ are k^{th} samples of the received long-training symbol 1 and 2, respectively.

The m^{th} samples beginning from the first sample of symbol L_1 is corrected by rotating at an angle of $\{-(m+192)\alpha\}$ because the first sample of symbol L_1 is separated 192 samples from the first sample of the frame. This rotation is implemented by using the well-known CORDIC algorithm [17]. The CORDIC algorithm is very convenient to implement on hardware, however, to get a high rotation accuracy, it requires a large number of iterations which has a large latency if implemented in software. The large number of cycles needed by the CORDIC algorithm as well as the high complexity of other algorithms for rotating samples explains why there are only few of previous software-based works proposing the CFO correction step in their receiver [6], [7]. We overcome this challenge by exploiting the multiple cores of AsAP2 architecture to rotate many samples in parallel. This increases overall throughput at the cost of using more processors. We will discuss further this issue in Section IV.

After CFO compensation step, the 16-sample GI field of each OFDM symbol are removed, then 64 data samples are transformed to the frequency domain by a 64-point FFT. Next, subcarriers in the frequency domain must be equalized to eliminate the effects of the communication channel. The channel's coefficients are estimated using information of two long-training symbols. Let $\widetilde{L}$ be the long-training symbol known by both the transmitter and receiver, and $\widetilde{L_1}$ and $\widetilde{L_2}$ be two long-training symbols rotated by the CFO compensation. The channel coefficient of the k^{th} subcarrier is estimated as follows:

$$H(k) = \frac{1}{2} \cdot \frac{\widetilde{L_1}(k) + \widetilde{L_2}(k)}{\widetilde{L}(k)} \tag{7}$$

This estimation of channel coefficients is used to equalize the corresponding subcarriers of all SIGNAL and DATA symbols. For each k^{th} subcarrier $S_m(k)$ of the m^{th} symbol, it is equalized by:

$$\widetilde{S_m}(k) = \frac{S_m(k)}{H(k)} \tag{8}$$

Above, we used two divisions to compute $H(k)$ and $\widetilde{S_m}(k)$. Since the division is slow, we should eliminate one at the equalization step by computing:

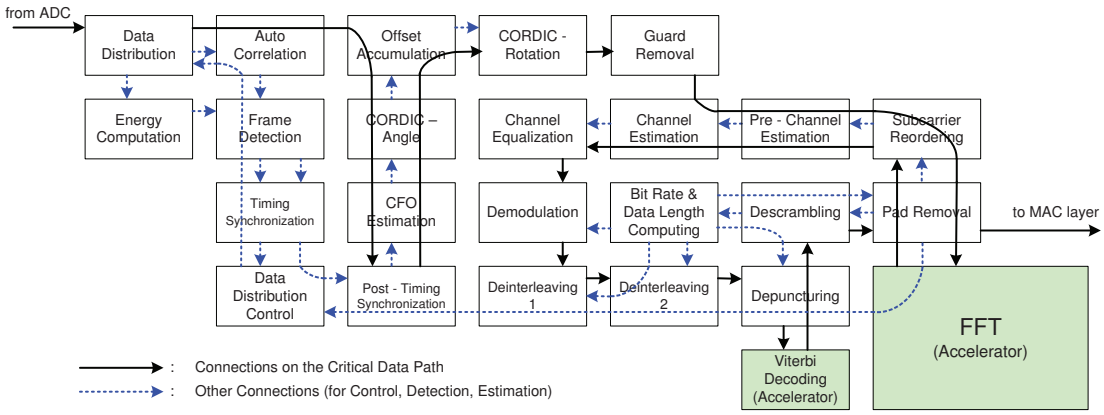

Fig. 5. Mapping diagram of the 802.11a baseband receiver on AsAP2. Each functional unit in Fig. 1 is mapped to one or many processors depending on its computational complexity. The Viterbi decoder and FFT are the configurable built-in accelerators

$$\widehat{S_m}(k) = S_m(k) \cdot C(k) \tag{9}$$

where $C(k)$ is estimated as:

$$C(k) = \frac{1}{H(k)} = \frac{2\widehat{L}(k)}{\widetilde{L_1}(k) + \widetilde{L_2}(k)} \tag{10}$$

to replace for estimation of $H(k)$ in Eq. 7.

After equalization, the subcarriers of OFDM symbols are demodulated into binary sequences and then these binary sequences are de-interleaved, de-punctured, decoded using Viterbi algorithm and descrambled. In the end, the final obtained binary sequence is sent to the Medium Access Control (MAC) layer. An important note is that the information from the SIGNAL symbol after decoded will be used to decide the modulation scheme (BPSK, QPSK, 16-QAM or 64-QAM), interleaving patterns and puncture pattern of the convolution code (1/2, 2/3 or 3/4) for DATA symbols corresponding to the supported bit rates: 6, 9, 12, 18, 24, 36, 48 or 54 Mbps.

III. Implementation of the 802.11a Baseband Receiver on AsAP2

A. Mapping 802.11a Receiver on the AsAP2 platform

Fig. 5 shows the mapping diagram of the 802.11a baseband receiver on AsAP2. Processors of the receiver are programmed using AsAP's assembly language. The following is a brief summary of each processor's task.

- **Data Distribution**: distributes input samples to other processors; each sample is represented by two 16-bit words (real and imaginary) in the 3.13 format.
- **Auto Correlation**: computes the auto-correlation function $P(n)$ given in Eq. 2.
- **Energy Computation**: computes the energy of each of 16 consecutive samples $Q(n)$ given in Eq. 3.
- **Frame Detection**: detects frame based on the condition of $|P(n)|^2 > 0.75Q(n)^2$ in 48 consecutive samples.
- **Timing Synchronization**: locates the first samples that satisfies $|P(n)|^2 < 0.15Q(n)^2$ after the frame was detected.
- **Post Timing Synchronization**: removes input samples until frame is synchronized.
- **Data Distribution Control**: controls the distribution of the samples.
- **CFO Estimation**: computes part $\{\sum_{k=0}^{63} L_2(k) \cdot L_1^*(k)\}$ of the carrier frequency offset given in Eq. 6.
- **CORDIC Angle**: computes angle α given in Eq. 6 using the CORDIC algorithm.
- **Offset Accumulation**: computes the angle $\{-(m+192)\alpha\}$ which is needed to rotate the m^{th} samples.
- **CORDIC Rotation**: rotates samples to compensate the carrier frequency offset.
- **Guard Removal**: removes 16 prefix samples of each 80-sample OFDM symbol.
- **FFT**: is FFT accelerator configured to perform a 64-point FFT on 64 samples of each OFDM symbol.
- **Subcarrier Reordering**: removes 12 null subcarriers and 4 pilots of each symbol. The remaining 48 subcarriers are then reordered as specified in the standard [1].
- **Pre-Channel Estimation**: computes the average value $\frac{1}{2} \cdot [\widetilde{L_1}(k) + \widetilde{L_2}(k)]$ for each of 48 subcarriers of two long-training symbols.
- **Channel Estimation**: computes the channel's coefficients $C(k)$ given in Eq. 10 using the division algorithm with a small lookup table introduced by Hung et al. [18].
- **Channel Equalization**: equalizes subcarriers $\widehat{S_m}(k)$ as given in Eq. 9.
- **De-modulation**: demaps each subcarrier back into a binary sequence. The number of bits representing each subcarrier depends on its modulation schemes: 1 bits, 2 bits, 4 bits and 6 bits for BPSK, QPSK, 16-QAM and 64-QAM, respectively.
- **Deinterleaving 1** and **Deinterleaving 2**: are two deinterleaving steps that reverse the interleaving steps from the transmitter.
- **Depuncturing**: inserts dummy bits into the locations removed by the convolution encoder of the transmitter for creating code rate 2/3 and 3/4 from the code rate 1/2.
- **Viterbi Decoding**: is a built-in accelerator used to decode the bit sequence using Viterbi algorithm.
- **Descrambling**: de-scrambles the decoded bit sequence using the generator polynomial $G(x) = x^7 + x^4 + 1$.
- **Bit Rate & Data Length Computation**: retrieves information about bit rate and data length from the decoded SIGNAL symbol. The bit rate tells information about the modulation scheme and code rate of the convolution code. This information is used to control the De-modulation, Interleaving 1, Interleaving 2 and Depuncturing processors. The data length tells the number of useful data bytes contained in the received frame.
- **Pad Removal**: removes garbage bits which include the Tail and Pad bits before sending the final bit sequence to the MAC layer.

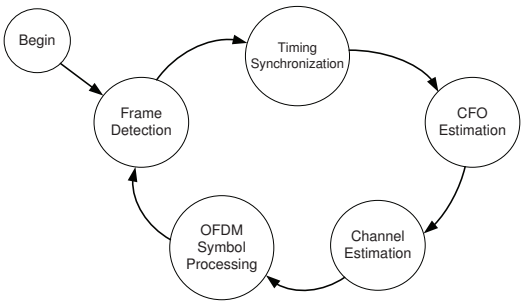

Fig. 6. Finite State Machine model of the receiver.

B. Critical Data Path and Reception of Multiple Frames

The dark solid lines in Fig. 5 show the connections between processors that are on the critical data path of the receiver. These processors processes all OFDM symbols in the form of a pipeline. The operation and execution time of these processors determine the throughput of the receiver. Other processors in the receiver are only briefly active for detection, synchronization (of frame) or estimation (of the carrier frequency offset and channel's coefficients); then they are forced to stop as soon as they finish their job. Consequently, these non-critical processors only add latency to the system and do not affect overall throughput.

After completion of a whole frame, in order for the system is able to receive another frame all stopped processors must be woken up. Therefore, the system is programmed to operate as a finite state machine (FSM) that is shown in Fig. 6.

In the beginning, the system operates in the Frame Detection state. The Data Distribution processor sends samples to both the Auto Correlation and Energy Computation processors for computing timing metrics $P(n)$ and $Q(n)$. These timing metrics are used to detect frame by the Frame Detection processor based on Eq. 4. After the frame has been detected, the system switches to the Timing Synchronization state, where timing metrics are used to synchronize timing by the Timing Synchronization processor based on Eq. 5.

Once the timing has been synchronized, the Data Distribution Control processor is informed and then it signals the Data Distribution processor to stop sending samples to the Auto Correlation and Energy Computation processors. Consequently, these processors are stopped and system is switched to the CFO Estimation state.

In the CFO Estimation state, the Post Timing Synchronization processor sends only 128 samples of two long training symbols to the CFO Estimation processor. Therefore, both CFO Estimation and CORDIC Angle processors are stopped after the frequency offset is estimated. Similarly, in the Channel Estimation state, the Subcarrier Reordering processor only sends 96 subcarriers of two long symbols to the Pre-Channel Estimation processor. Eventually, both the Pre-Channel Estimation and Channel Estimation processors also stop working after computing all 48 coefficients of channel.

Once the channel has been estimated, the system is in the OFDM Symbol Processing state where all OFDM symbols (including the SIGNAL and all DATA symbols) are processed by the processors on the critical path. After the whole frame is received and processed, the Pad Removal processor tells the Data Distribution Control processor to allow the Data Distribution processor resending samples of the new frame (if any) to the Auto Correlation and Energy Computation processors. Then it also resets the Subcarrier Reordering and De-scrambling processors. Now, the system has returned to the Frame Detection state and is ready to receive another frame.

The architecture of AsAP2 is extremely flexible that allows to program the receiver for complying a dynamically complicated finite state machine as described above. Each processor has two input FIFOs; we can use one for data buffering and the another one for control words. Moreover, a processor can be forced to sleep (stop executing) by stopping sending data to it. It then will be woken up once there are any data sent to its FIFOs.

TABLE I
EXECUTION TIME OF ALL PROCESSORS FOR PROCESSING ONE OFDM SYMBOL WITHOUT STALL FOR WAITING INPUT AND OUTPUT

Processor Name	# of Inputs Needed	# of Outputs Needed	Exe. Time (cycles)	On the Critical Path?
Data Distribution	**80**	**80**	**320**	✓
Energy Computation	80	80	1360	-
Auto-correlation	80	80	4960	-
Frame Detection	80 + 80	80	1200	-
Timing Synch.	80 + 80	1	1360	-
Data Distr. Control	≤ 1	≤ 1	5	-
Post-Timing Synch.	**80**	**80**	**240**	✓
CFO Estimation	128	1	989	-
CORDIC Angle	1	1	183	-
Offset Accumulator	1	80	560	-
CORDIC Rotation	**80 + 80**	**80**	**15120**	✓
Guard Removal	**80**	**64**	**176**	✓
64-point FFT	**64**	**64**	**205**	✓
Subcarrier Reorder	**64**	**48**	**1018**	✓
Pre-Channel Estimation	48	48	1832	-
Channel Estimation	48	48	6960	-
Channel Equalization	**48**	**48**	**1488**	✓
Demodulation	**48**	**288**	**2352**	✓
Deinterleaving 1	**288**	**288**	**864**	✓
Deinterleaving 2	**288**	**288**	**1130**	✓
Depuncturing	**288**	**432**	**576**	✓
Viterbi Decoding	**432**	**216**	**2376**	✓
BR & DL Computing	24	2	95	-
Descrambling	**216**	**216**	**2160**	✓
Pad Removal	**216**	**≤ 216**	**648**	✓

IV. Evaluation and Improvement of the Receiver's Throughput

To evaluate the overall throughput of the receiver, our approach method is considering the execution time of each separate processor first. Then, based on this information, we can determine where the bottleneck occurs that is needed to improve for getting a higher rate.

A. Throughput Evaluation

The execution time of each processor is separately evaluated while it runs alone without connected with other processors. This evaluation shows the pure execution time of each processor without any stall on input and on output caused from other processors. The execution time (in clock cycles) of each processor is counted from the first input word received to the last output word sent. Fairly, we evaluate the execution time of each processor while processing one OFDM symbol that is required to be in 4 μs to obtain full rate.

Table I shows the execution time of each processor when processing one OFDM symbol in the 54 Mbps mode, the slowest operational mode. Other seven modes are much simpler that certainly have smaller number of execution cycles; therefore considering them is unnecessary here. Column 2 and 3 show the number of inputs and outputs processed by a processor in the duration of one OFDM symbol. These inputs and outputs might be samples (represented by 2 words for real and imaginary parts) in the time domain (before FFT), subcarriers (2 words) in the frequency domain (after FFT), bits (after de-modulation) or control words. The "+" sign indicates processor which requires inputs to both FIFOs. Note that, although some other processors have inputs to both FIFOs, only one FIFO is data while other FIFO contains control words which are only used to control the states of the FSM; therefore they almost do not affect the execution time of these processors.

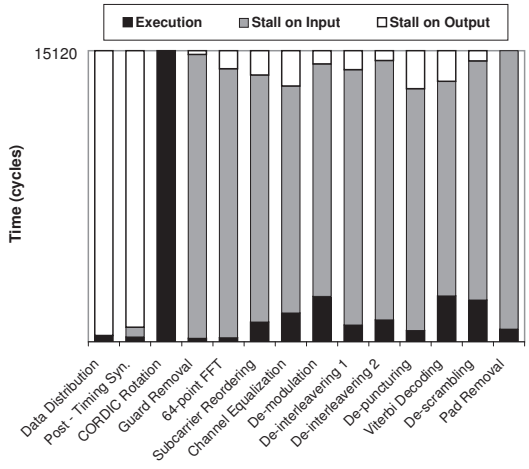

Fig. 7. The overall activity of processors on the critical data path when processing one OFDM symbol.

As mentioned in the Subsection III-B, the throughput of the receiver will be mainly determined by the slowest processor on the critical path. Other processors just contribute to the latency but not the overall throughput. The execution time of processors shown in Table I is ideal, *i.e.* without taking into account the stalled time while waiting to receive an input or to send an output. The overall behavior of processors on the critical path including both their execution and stalled time is shown in Fig. 7.

The CORDIC Rotation processor used for CFO compensation is shown to be the system bottleneck; it is nearly seven times slower than the other processors. It is always busy executing and forces other processors on the critical path to stall either on output while sending data or on input while receiving data. However, the total of execution time and stall time of each processor is fixed and is equal to the execution time of the CORDIC processor (15120 cycles). As a result all OFDM symbols are processed by the sequence of processors on the critical path in a way that is similar to a pipeline. Therefore, to get a real-time throughput of 54 Mbps, each processor has to process one OFDM symbol (216 bits) in 4 μs that requires them to operate at 3.78 GHz. This clock frequency is not supported by the AsAP2 chip. In order for the receiver can sustain real-time requirement at a lower clock frequency, we have to reduce the execution time of the angle rotation.

B. Optimizing Throughput

There are some good algorithms for angle rotation in the literature [17], [19]; however, all of them were originally proposed for hardware implementation. Implementation of these algorithms in software is either so complex or requires many iterations to get high accuracy. We choose the CORDIC algorithm because its iterative method is most simple. In our best effort, however, its implementation requires 189 cycles for completing a rotation (that is 15120 cycles to rotate 80 samples of an OFDM symbol). In a multiprocessing platform as AsAP2, the most convenient solution for speeding this up is by using many CORDIC processors to rotate samples in parallel.

The Fig. 8 describes a traditional solution for making three CORDIC operations in parallel. One processor is used to distribute samples to three CORDIC processors for rotation. The outputs from these CORDIC processors are collected by some other processors that then form a sequence of rotated samples to be sent to the Guard Removal processor. The big drawback of this solution is that we waste many processors for distribution and collection purposes only.

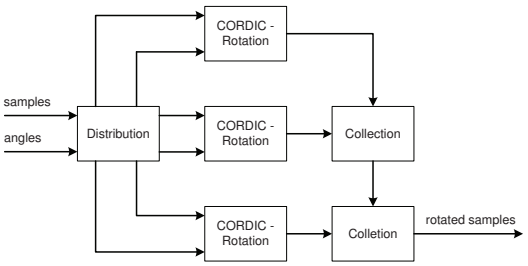

Fig. 8. Parallel operation of three CORDIC processors by the traditional solution requires at least six processors.

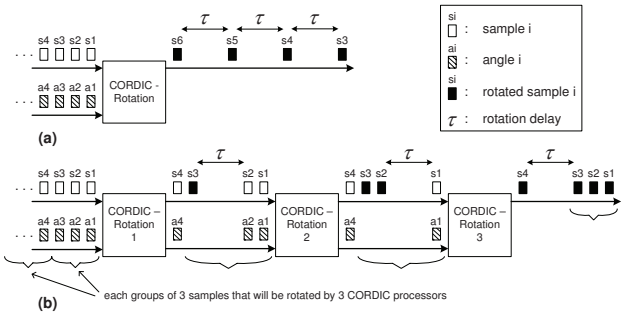

Fig. 9. Solution of using only three processors for performing three CORDIC rotations in parallel. a) Operation illustration of using just one CORDIC processor. b) Illustration of three CORDIC processors.

Generally, it requires at least N processors for supporting N CORDIC processors, that is more than $2N$ processors in total.

A novel solution of using exactly N processors for performing N CORDIC rotations in parallel is shown in Fig. 9. Fig. 9(b) illustrates the operation of three CORDIC processors. Input samples are processed in groups of three samples each. Considering the first group of three samples 1, 2 and 3. The first CORDIC processor passes the first and second samples and angles to the second processor, then rotates the third sample by the third angle. The second processor passes the first sample and angle to the third processor and then it rotates the second sample by the second angle. Finally, the third processor rotates the first sample by the first angle. Because the rotation delay of each processor is so high compared to the passing time, three samples are almost simultaneously rotated by three processors. After finishing rotation of the first three samples, the processors continue to process for group of three samples 4,5 and 6; and so on. Consequently, when looking at the output of the final CORDIC processor, their overall throughput is three times faster than that of using just one CORDIC as illustrated in Fig. 9(a) with a small overhead due to passing time of samples through all three processors..

This idea is straightforward to scale for performing N CORDIC

TABLE II
PERFORMANCE OF THE RECEIVER CORRESPONDING TO THE NUMBER OF PARALLEL CORDIC PROCESSORS

Number of CORDIC processors	Exe. time per symbol (cycles)	Output throughput (bits / cycle)	Frequency to obtain 54 Mbps
1	15120	0.014	3.78 GHz *
2	7684	0.028	1.93 GHz *
3	5262	0.041	1.32 GHz *
4	3961	0.055	990 MHz
5	3216	0.067	800 MHz
6	2718	0.080	680 MHz
7	**2376**	**0.091**	**590 MHz**

*: *not supported frequency*

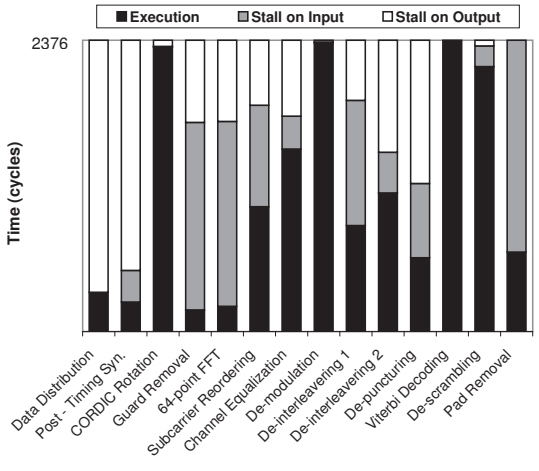

Fig. 10. The overall activity of processors with seven CORDIC Rotation processors in parallel for one 4 μsec-OFDM symbol

operations in parallel by using only N processors. The receiver's performance corresponding with the number of parallel CORDIC Rotation processors is shown in Table II. As shown, the throughput increases approximately proportional with the number of parallel CORDIC Rotation processors. However, the throughput does not change when the number of parallel CORDIC Rotation processors is larger than seven. This happens because with more than seven CORDIC Rotation processors, their execution time for processing one OFDM symbol is now less than that of the Viterbi processor. The Viterbi processor becomes the receiver's bottleneck and therefore determines the throughput.

Fig. 10 shows the overall receiver activity when using seven CORDIC Rotation processors in parallel. The total processing time for one OFDM symbol of each processors including execution time and stall times now equals to the execution time of the Viterbi processor which is 2376 cycles. The Viterbi processor is a built-in accelerator on AsAP2, so no further improvement of performance is possible by software programming. Consequently, the receiver can obtain a 54 Mbps throughput at clock frequency of 590 MHz.

V. Conclusion and Future Work

In this paper, we have presented the design of a complete baseband receiver compliant with the IEEE 802.11a standard and its software implementation on the AsAP2 platform. The processors are programmed to realize a dynamic FSM model that maximizes the throughput and minimizes the latency while ensuring the continuous reception of multiple frames. The high flexibility of the AsAP2 platform allow us easily mapping the system to obtain a real-time throughput of 54 Mbps while operating at only 590 MHz. Our receiver was fully tested on the real AsAP2 chip and it works accurately compared with the result of a correct Matlab model.

If processors are configured to run at their maximum frequency of 1.2 GHz, our receiver can achieve a bit rate up to 110 Mbps. Clearly, the AsAP2 platform has potential for implementing other wireless applications that require much higher throughput. Its high performance is achieved due to the task-level parallelism on as many small processors as possible rather than using data-level parallelism as on other SIMD or VLIW platforms.

Currently, we are focusing on optimizing the configurable accelerators and upgrading the AsAP platform for mapping new wireless standards such as 802.11n WLAN and 802.16e WiMAX. We expect that, in the future, the AsAP platform will be able to have more than 1000 small processors and accelerators that is highly feasible and efficient to implement all complex wireless baseband applications in software.

Acknowledgments

The authors would like to thank members of VCL for discussion and assistance. This work was supported by IntellaSys, a VEF Fellowship, SRC GRC Grant 1598 and CSR Grant 1659, ST Microelectronics, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, Intel, and S Machines.

References

[1] 802.11a Standard, "Wireless lan medium access control (mac) and physical layer (phy) specifications: High-speed physical layer in the 5 ghz band," Tech. Rep., IEEE Computer Society, 1999.

[2] M. J. Meeuwsen et al., "A full-rate software implementation of an ieee 802.11a compliant digital baseband transmitter," in *IEEE Workshop on Signal Processing Systems, SiPS*, Oct 2004.

[3] Y. Tang et al., "Optimized software implementation of full-rate ieee 802.11 a compliant digital baseband transmitter on digital signal processing," in *Global Telecommunications Conference, GLOBECOM*, 2005.

[4] Y. Lin et al., "SODA: A high-performance DSP architecture for software-defined radio," *IEEE MICRO*, vol. 27, no. 1, pp. 114–123, Feb. 2007.

[5] M. F. Tariq et al., "Development of an ofdm based high speed wireless lan platform usingthe ti c6x dsp," in *Int. Conference on Communications, ICC*, Apr 2002, pp. 522–526.

[6] J. D. Bakker and F. C. Schoute, "Lart: Design and implementation of a experimental wireless platform," in *IEEE Vehicular Technology Conference*, Sep 2000, pp. 1460–1466.

[7] S. Eberli et al., "An ieee 802.11a baseband receiver implementation on an application specific processor," in *Midwest Symposium on Circuits and Systems, MWSCAS*, Aug 2007, pp. 1324–1327.

[8] E. Tell et al., "A programmable dsp core for baseband processing," in *IEEE-NEWCAS Conference*, Jun 2005, pp. 403– 406.

[9] A. Niktash et al., "A case study of performing ofdm kernels on a novel reconfigurable dsp architecture," in *Military Communications Conference, MILCOM*, Oct 2005, pp. 1813–1818.

[10] K. Akabane et al., "Design and performance evaluation of ieee 802.11 a sdr software implemented on a reconfigurable processor," *IEICE Transactions on Communications*, pp. 4163–4169, Nov 2005.

[11] D. Truong et al., "A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling," in *Symposium on VLSI Circuits*, June 2008.

[12] Z. Yu et al., "Asap: An asynchronous array of simple processors," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 43, no. 3, pp. 695–705, Mar. 2008.

[13] T. M. Schmidl and D. C. Cox, "Rubust frequency and timing synchronization for ofdm," *IEEE Transactions on Communications*, vol. 45, pp. 1613–1621, Dec. 1997.

[14] V. Jiménez et al., "Design and implementation of synchronization and agc for ofdm-based wlan receivers," *IEEE Transactions on Consumer Electronics*, vol. 50, pp. 1016–1025, Nov. 2004.

[15] H. Tang et al., "Synchronization schemes for packet ofdm system," in *Intl. Conference on Communications, ICC*, May 2003, vol. 5, pp. 3346–3350.

[16] E. Sourour et al., "Frequency offset estimation and correction in the ieee 802.11a wlan," *IEEE Vehicular Technology Conference*, vol. 7, pp. 4923–4927, Sept. 2004.

[17] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *ACM/SIGDA Intl Symposium on FPGA*, 1998, number 6, pp. 191–200.

[18] P. Hung et al., "Fast division algorithm with a small lookup table," in *IEEE Asilomar Conference on Signals, Systems, and Computers*, Oct. 1999, vol. 2, pp. 1465–1468.

[19] Y. Song and B. Kim, "A 16b quadrature direct digital frequency synthesizer using interpolative angle rotation algorithm," in *Symposium on VLSI Circuits*, June 2002, pp. 146–147.