OPTIMIZED LOW-POWER SYNCHRONIZER DESIGN FOR THE IEEE 802.11a STANDARD

Miloš Krstić, Alfonso Troya, Koushik Maharatna, Eckhard Grass

IHP
Im Technologiepark 25, 15236 Frankfurt (Oder), Germany
E-mail: {krstic, troya, maharatna, grass}@ihp-microelectronics.com

ABSTRACT

In this paper the authors propose a low-power synchronizer design for the IEEE 802.11a standard capable to estimate frequency offsets in the range ±468 kHz (80 ppm @ 5.8GHz) with very simple and effective frame detection and timing synchronization. The core area of the design after layout is 13 mm^2, including the CORDIC and FFT processors, with a total estimated power consumption of 140 mW.

1. INTRODUCTION

OFDM signals are very sensitive to the synchronizer performance, mainly because the different sub-carriers overlap their respective spectra. The synchronizer is the block responsible for detecting the incoming frame and to estimate and correct for the possible frequency offsets. It also decides the starting point from which on the different OFDM symbols will be fed into the FFT block. To carry out all of these operations, the IEEE 802.11a standard defines a specific periodic symbol structure known as *preamble symbols*, that are appended at the very beginning of each transmitted frame [1].

During synchronization the following operations have to be carried out: frame detection, carrier frequency offset estimation, symbol timing estimation, extraction of the reference channel and data reordering. The theory of synchronization algorithms for this particular application is well known and will not be described in detail in this paper. Some theoretical background on the different operations involved in the synchronization procedure was presented in [2]. Nevertheless, the structure described there was primarily a simulation model. To realize an implementation friendly hardware architecture, several blocks needed to be optimized. Furthermore, the solution proposed in [2] introduced a latency of more than 4 μs into the design, thus making necessary the use of temporary data storage, which increased the core area of the final design. In the present work we focus on the critical hardware issues and accordingly, an optimized design for the synchronizer is presented. The paper is structured as follows: in Section 2, the synchronizer data path structure is described in detail. In Section 3, the optimization of the most power consuming blocks in this implementation is investigated and innovative solutions are proposed. Finally, in Section 4 some conclusions are derived.

2. SYNCHRONIZER DATA PATH STRUCTURE

The general structure of the synchronizer is shown in Figure 1. Two mutually exclusive operations are present in the system: tracking for the training sequences and processing of the observed symbols. This fact can be used to obtain a power efficient synchronizer design by applying clock gating. Thus the whole structure was split into three clock domains and two mutually exclusive paths: *tracking data path* and *processing data path*.

2.1. Tracking data path

The main function of the tracking data path is to detect an incoming frame by searching for the periodic structure of the preamble symbols and to estimate the carrier frequency offset. The constituent blocks are (see Figure 1):

Autocorrelators. In our design two autocorrelators, having lengths of 64 and 16 samples, respectively, are used. The autocorrelator with length 64 is used for the frame detection and to get a *fine* estimation of the carrier frequency offset (parameter α). The latter (length 16) is used to provide a *coarse* estimation of the carrier frequency offset (parameter β). The length of the autocorrelator determines the range of frequency offsets, which can be estimated. Thus, in the present case α will be in the range [-0.5, +0.5), and β in the range [-2.0, +2.0), α and β being normalized values with respect to the sub-carrier spacing Δf (Δf = 312.5 kHz in the IEEE 802.11a). The final estimated value of the frequency offset, ε, is a non-linear combination of α and β [3]. Figure 2 shows the output $|J_F(k)|^2$ of the long autocorrelator. Two plateaus of

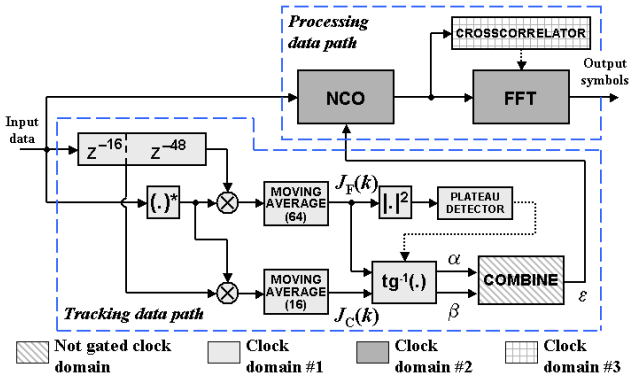

Figure 1. General scheme of the Synchronizer.

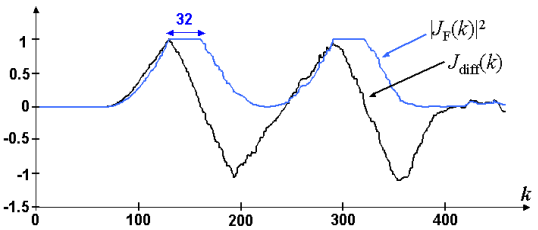

Figure 2. Signals involved in the plateau detection algorithm.

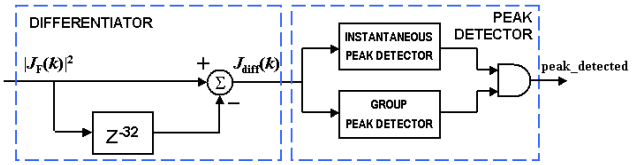

Figure 3. Scheme of the *plateau detector*.

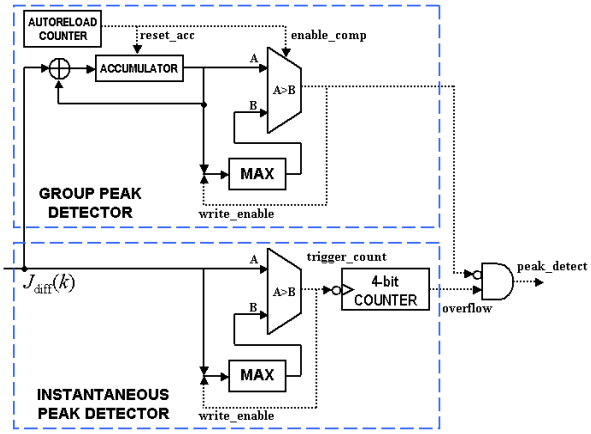

Figure 4. Implementation of the peak detector.

length 32 samples each can be distinguished. The frame detection algorithm is based on the detection of the starting point of the first plateau.

Plateau detector. This block is built with a *differentiator* and a *peak detector*. Using the differentiator shown in Figure 3 it is possible to obtain a signal with an absolute maximum at the point where the first plateau starts. This signal is labeled $J_{diff}(k)$ in Figure 2. The next step is the detection of the absolute maximum in $J_{diff}(k)$, making use of the peak detector shown in Figure 3. This peak detector itself is divided into two blocks, namely *group peak detector* and *instantaneous peak detector*.

The instantaneous peak detector is composed of a comparator and a counter (see Figure 4). The present sample $J_{diff}(k)$ coming from the differentiator is compared with the last recorded maximum J_{max}. As long as the sample $J_{diff}(k)$ is bigger than J_{max}, the register storing J_{max} will be updated with the new sample $J_{diff}(k)$ as the latest maximum and the counter will be reset. If $J_{diff}(k)$ is smaller than or equal to J_{max}, the counter will be incremented. If this situation remains until the counter counts through its full range, the instantaneous peak detector will generate a signal stating that a relative peak was found inside the counting scope of the counter.

The group peak detector is used to detect the falling edges in $J_{diff}(k)$, and its main component is also a comparison block. There, the input signal is accumulated in groups of six samples (6-tuples) and the present group is compared with the previous one. If it is smaller, it means that the falling slope has started. If the group peak detector finds a falling edge at the same time as the instantaneous peak detector finds a relative peak, then the detected peak is actually an absolute peak.

Due to the selected implementation of the plateau detector, the first plateau is detected 16 samples after its actual starting point, i.e. the detection will occur in the middle of the plateau. Furthermore, in order to avoid the problem of false frame detections resulting from noise, a threshold is applied to the signal $|J_F(k)|^2$ prior to the plateau detection block.

Arctangent. The values of α and β will be obtained after calculating the phase of the complex samples $J_F(k)$ and $J_C(k)$ respectively, at that time *k* where the peak detector established the beginning of the frame (depicted in Figure 1). The *arctangent* calculation ($tg^{-1}(.)$) can be efficiently realized by using the CORDIC algorithm working in the *vectoring mode*.

2.2. Processing data path

The activity of the processing data path starts when the frame is detected and the estimated value for ε is available. This part of the synchronizer performs the carrier frequency error correction, estimates the symbol timing and obtains the reference channel estimation. This consists of the following blocks:

Numerically Controlled Oscillator (NCO): The correction of the frequency offset is carried out by an NCO, which is implemented using the CORDIC algorithm again, this time operating in the *rotational mode*. The NCO is activated once the estimation of the frequency offset ε is available and operates until the end of the frame

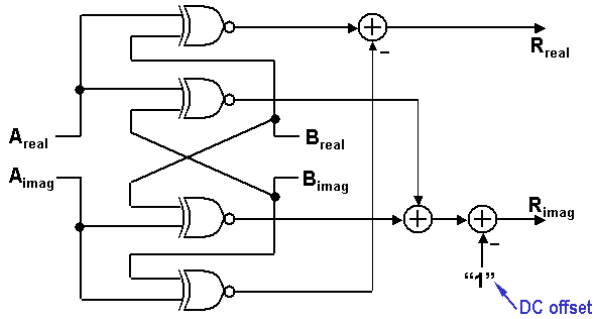

Figure 5. XNOR-based complex multiplier.

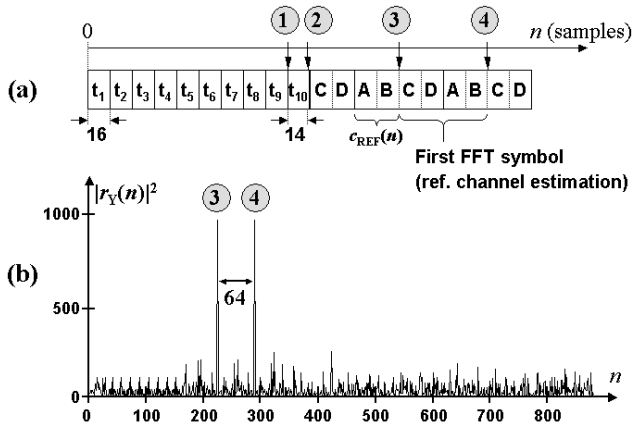

Figure 6. Timing scheduling at the synchronizer.

is detected.

Crosscorrelator: After the frequency offset correction, the next processing task is the symbol timing estimation. The symbol timing is obtained by exploiting the direct knowledge of the *long preamble symbols*, and is based on a *crosscorrelation*. The main purpose of the crosscorrelator is to compare the input frame with a reference signal, which is directly obtained from the long preamble symbol. This reference is the sequence $c_{REF}(n)$ (fields **AB** in Figure 6.a) and is only 32 samples long, as this is the shortest length necessary in order to get meaningful results at the output of the crosscorrelator. This output is represented in Figure 6.b showing two clear peaks (marked as **3** and **4**). Both peaks will occur when the portions of the long preamble symbols, which are identical to $c_{REF}(0…31)$, are inside the crosscorrelator. For our purpose it is enough to detect the first peak by setting a certain threshold at the output of the crosscorrelator. Once this peak is detected, the crosscorrelator is deactivated.

As the position of the peaks **3** and **4** relative to the preamble symbols is known beforehand (Figure 6.a), the symbol timing is resolved directly. In addition, the time instant **1** shown in Figure 6.a would be the ideal instant at which the synchronizer has the estimation of ε available for the NCO. Furthermore, the time from **1** to **2**, i.e. 14 samples @ 20MHz, is the latency introduced by the NCO. Note that the crosscorrelator can only be applied once the input frame is free of any frequency offset.

FFT processor. Immediately after the symbol timing is found, the next 64 samples (**CDAB** field in Figure 6.a) are fed into the 64-point FFT in order to get the *reference channel estimation*. In the IEEE 802.11a standard the long preamble symbol is defined as the sequence **ABCD**, i.e. in our case a cyclic delay of 32 samples is introduced into a sequence of 64 samples. Therefore, the resulting sequence after FFT calculation has to be multiplied by $(-1)^k$, k being the frequency variable, in order to correct for the remaining linear phase.

It is to be noted that the standard defines two long preamble symbols **ABCD** in order to obtain initially two raw channel estimations and calculate the final reference channel estimation by averaging them and thus, a 3 dB improvement in the SNR would be possible. However, in practice this is difficult to implement, considering the overall latency constraints imposed by the MAC layer.

For the data symbols coming after the preamble, two further operations have to be performed inside the FFT block. Prior to the FFT calculation, the data symbols will go through a *cyclic prefix* extraction block, since for each data symbol 80 samples are expected but only the last 64 are fed into the FFT. The last operation is the channel reordering, i.e. after FFT calculation the output samples are delivered in serial form according to the natural order. For further processing this order has to be changed. Note that the phase correction by the sequence $(-1)^k$ after FFT is no longer necessary in the data symbols. The FFT will then operate until the end of the frame, thus being part of the same clock domain as the NCO.

3. BLOCK OPTIMIZATION AND RESULTS

In the previous Section it was pointed out that a first mechanism for power saving was the division of the whole architecture into different clock domains. A second level of power reduction is achieved by optimizing the different modules of the synchronizer.

In our architecture we separated the NCO and the arctangent blocks since the control mechanism is much simpler compared to the use of a single full CORDIC. Here, the silicon area is 33% more, but the power is reduced by 54% owing to the clock gating. The CORDIC processors designed for these purposes have been optimized to reach the final angle in an adaptive way and thereby executing a minimum number of iteration steps. The implementation is based on the virtually scaling-free adaptive CORDIC proposed in [4].

The complex crosscorrelator is a challenging design problem because of its computation complexity (it requires a large number of complex multipliers) and subsequent large silicon area. In our proposal, a simplified complex multiplier architecture based on XNOR 1-bit multipliers is

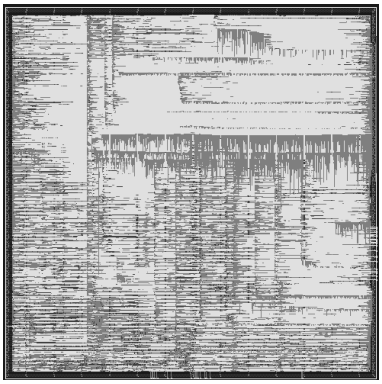

Figure 7. Core layout of the Synchronizer.

Component	Cell area (mm^2)	Power (mW)
FFT	3.37	140*
NCO	0.78	6.6
Arctangent	0.8	7.2
64-tap Delay line	0.49	110
Moving Average (16 & 64)	0.49	115
Square magnitude	0.12	3.8
Plateau detector	0.23	8
α and β combining	0.06	0.8
Crosscorrelator	0.1	1

* Measured power consumption after fabrication as a discrete component is 84 mW @ 2.5 V, 20 MHz.

Table 1. List of components of the Synchronizer.

used (Figure 5). Instead of multiplying N-bit complex numbers, the XNOR multiplier performs the multiplication of the sign bits of the complex input values only. In addition, the XNOR gates are replaced by NOT gates since one of the inputs is fixed and known beforehand.

The design of the FFT processor is based on the radix-8 FFT, which we found optimal for the implementation of the 64-point FFT. The processor requires 49 non-trivial complex multiplications to perform the transformation with a latency of 23 cycles parallel-to-parallel. More details on the implementation may be found in [5].

The described synchronizer has been synthesized in our in-house 0.25µm BiCMOS technology. After synthesis, the power analysis made by Synopsys resulted in power figures of 140 and 175 mW, with and without clock gating, respectively. From our experience we expect even a greater difference between those two architectures. The core layout area is 13 mm^2 (Figure 7). The whole structure introduces a latency of 3.9 µs counted from the instant when the frame is detected. This value is less than one OFDM symbol period (4 µs), meaning that no additional storage for the input samples is necessary inside the synchronizer.

In our design, a wide range of frequency offsets can be estimated (±80ppm) using only two autocorrelators, and the output of one of those is used in the frame detection mechanism. This provides a big core area reduction in comparison with other proposed solutions, as in [6], where the range of estimated frequency offsets is ±40ppm and three autocorrelators are used for the frame detection, but only two of them for the frequency offset estimation.

In Table 1, a list of the main components used in the synchronizer is provided, together with their respective cell area and estimated power consumption.

4. CONCLUSIONS

The hardware implementation of a novel low-power synchronizer for the IEEE 802.11a standard has been described in this paper. The selected integration strategy along with a distributed control enables the division of the full structure into different clock domains and simplifies the power management. Furthermore, this design reduces the overall latency avoiding any additional data buffering.

The proposed architecture has been synthesized using our in-house 0.25µm BiCMOS technology. The layout was done with some extra routing channels in-between the standard cell rows, resulting in a core area of 13 mm^2 and an estimated power consumption of 140 mW when clock gating was used.

To our knowledge, this is the least power consuming synchronizer architecture for the IEEE 802.11a standard reported so far.

5. REFERENCES

[1] IEEE P802.11a/D7.0, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High Speed Physical Layer in the 5 GHz Band*, Piscataway, NJ, 1999.

[2] A. Troya, K. Maharatna, M. Krstic, E. Grass, R. Kraemer, "OFDM Synchronizer Implementation for an IEEE802.11a Compliant Modem", *Proc. IASTED International Conference on Wireless and Optical Communications*, Banff, Canada, pp. 152-157, July 2002.

[3] Yun Chiu et al., OFDM Receiver Design, Final Report 12/12/2000. *Downloaded from http://bwrc.eecs.berkeley.edu*.

[4] K. Maharatna, *CORDIC based signal processors for biomedical applications*, Ph.D. Dissertation, Jadavpur University, India, 2002.

[5] E. Grass et al., "On the Single-Chip Implementation of a Hiperlan/2 and IEEE 802.11a Capable Modem", *IEEE Personal Communications*, vol. 8, no. 6, pp. 48-57, Dec. 2001.

[6] L. Schwoerer, H. Wirz, "VLSI Implementation of IEEE 802.11a Physical Layer", *Proc. 6th Int'l. OFDM Workshop*, Hamburg, Germany, pp. 28.1 – 28.4, Sept. 2001.