AALBORG UNIVERSITY

Department of Electronic Systems

Comparison of OFDM Acquisition Methods and Their Architectural Design

Marko Zorica

Master Thesis / 2010-2011

Table of contents

1. Introduction

Wireless communication systems can be found all around the world today. Their enormous growth in last few decades greatly changed our life and today it is impossible to imagine world and people's life without them. One of first well known inventions in wireless communications was telegraph and it used sound and light signals to transmit the message. Discovery of electromagnetism and radio waves was crucial to development of wireless communications. Today, there is a huge list of systems which utilizes radio waves. Without radio and television broadcasting systems it is almost impossible to imagine our life. Mobile phones are actually one of fundamental tools for business today, and for communication as well. WLAN (Wireless Local Area Network) networks gradually replacing wired networks in universities, campuses, companies and even in homes. WLAN networks are connecting devices (e.g. PC, laptop) with access point, which usually provides them access to the internet.

In wireless communication there are two main types of systems considering the way of data transmission. First one for data transfer uses one carrier and is modulated by signal which contains information. Such system is called single-carrier system. It is mostly used in AM and FM radio broadcasting systems, in analog television broadcasting systems, in old generations of cellular systems (1G, 2G and 3G) and many others. First generation of cellular wireless systems (used mostly for mobile communication systems) used analog signals for data transmission and their speed was 9.2 kbit/s. First generation was replaced by second generation of cellular wireless systems. It was first generation of cellular networks which used digital signal for data transmission. To improve quality of the signal, speech coding was also introduced what was not possible in analog system. Besides, this generation provided some additional services such as caller ID. In GSM system, the best known 2G system, speed was 20 to 150 kbit/s. Third generation of cellular wireless systems is called 3G and has data rates 1-2 Mbit/s. 3G generation of cellular wireless systems can provide much more services than 2G systems, such as Internet access and video mobile calls, and as mentioned above, data rate is several times higher. However, during the huge expansion of Internet and technology in general, demands for services and data rates were increased. The most recent generation of cellular wireless systems is 4G. This system is still developing, but once when it would be fully developed and adopted there are expected data rates between 100 Mbit/s and 1Gbit/s. The most known 4G system today is IEEE 802.16, known also as WiMAX and is used for mobile internet access.

Systems of fourth generation for data transmission use the second one system which is called multi-carrier system. This system uses two or more carriers for data transfer, instead of one as aforementioned systems. Each of those carriers is modulated by one conventional digital modulation such as BPSK or QAM. The most known such system is called OFDM (Orthogonal Frequency Division Multiplexing).

The main imperfections of single-carrier systems are its vulnerability to fading caused by multipath propagation and to changes of channel frequency response. To cope with those problems, we can use multi-carrier system, e.g. OFDM. OFDM occupies almost same bandwidth as single-carrier system, but can cope with aforementioned problems more effective. Also, OFDM has a higher spectral efficiency than single-carrier systems. To increase robustness against intersymbol interference to each OFDM symbol is added a guard interval. It is filled with cyclic prefix which is a copy of last part of OFDM symbol, as shown in figure 1. It will be explained in more detail in further text.

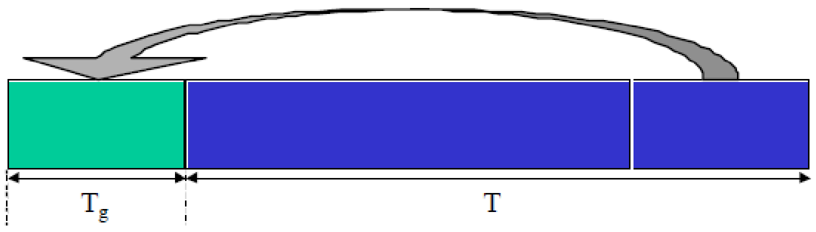

Figure 1 – OFDM symbol and cyclic prefix

Beside these advantages, OFDM also has some disadvantages. OFDM is very sensitive to carrier frequency offset and to sampling frequency offset. Also, it has a high PAPR (Peak-to-peak average ratio), and is vulnerable to synchronization problems (e.g. frame start detection). All of these problems can be solved using a certain algorithms which are implemented in the receiver. In this thesis focus will be on synchronization and coarse carrier frequency offset. Since there is several existing algorithms to solve this problem, in this thesis will be analyzed some of these methods to see which method is the best and most reliable. There are two main types of methods to estimate frequency offset and to detect start of the OFDM symbol. One of them is guard interval based (GIB) method and it is used in broadcasting systems such as DVB-T (Digital Video Broadcasting – Terrestrial). Second one is preamble based method which is used systems like WLAN systems and in fourth generation of cellular networks (4G systems). Difference between these two methods will be explained in next chapter.

2. OFDM

In this chapter will be described some OFDM basics which are necessary for understanding of problem with which this thesis deals. As mentioned in first chapter, OFDM is a system which utilizes two or more carriers for data transmission. In FDM (Frequency Division Multiplexing) systems spectra of each subcarrier does not overlap with spectrum of adjacent (or some other) subcarrier. Also, there are guard bands between each subcarriers. Spectra of OFDM subcarriers are overlapped but still there is no Inter-carrier Interference (ICI) because of orthogonality property. Spectrum of OFDM signal is shown on figure 2.1. It is obvious that OFDM system will occupy considerable less bandwidth than FDM system.

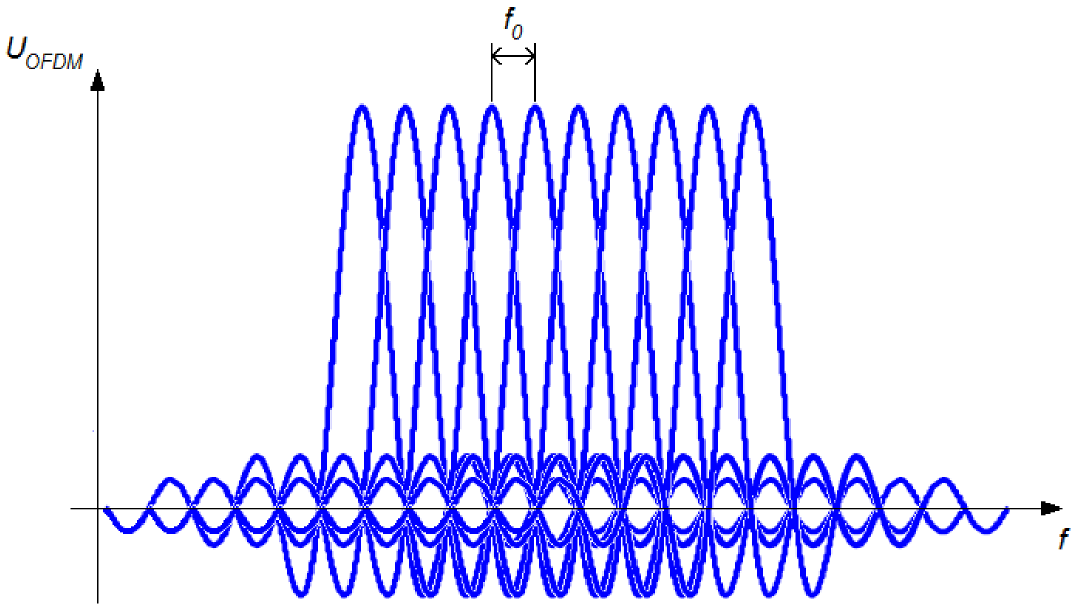

Figure 2.1. – OFDM signal spectrum

Carriers will be orthogonal if the center of their spectrum is located in zeros of other's carriers spectrum. Condition to be meet to achieve orthogonality is that carrier spectra must be spaced for frequency $f_0$or integer multiplier of f_0. f_0 can be calculated as $f_0 = 1/T_0$ where T_0 is a duration of one symbol on each modulated carrier, and also duration of OFDM symbol. As mentioned in chapter 1, each subcarrier in OFDM system is modulated by one of standard digital modulation such as QAM or PSK.

2.1. Technique of obtaining OFDM signal

In figure 2.2. is shown block scheme of typical OFDM transmitter.

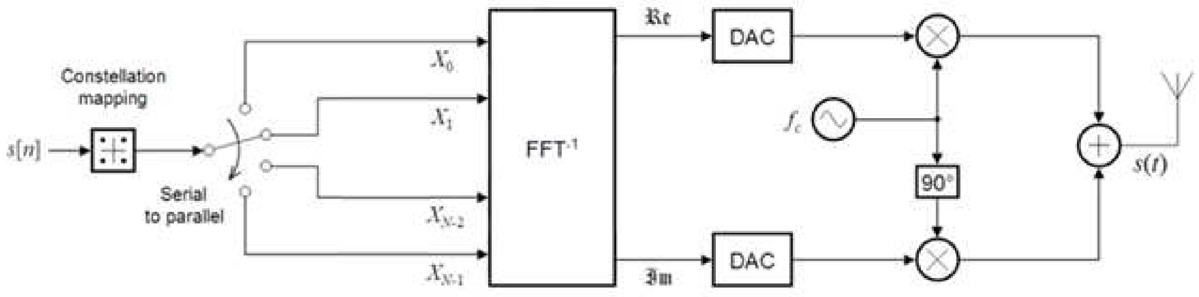

Figure 2.2. – Block scheme of OFDM transmitter

Serial data stream modulated by QAM or PSK modulation is converted in parallel data stream. One symbol in each data stream is brought on the IFFT block. After performing Inverse Fast Fourier Transform, on IFFT output is digital signal which has same number of samples as number of IFFT bins, and it is actually one OFDM symbol. Duration of OFDM symbol is determined by number of IFFT bins and sampling period T_s .OFDM symbol can be described by

$$s(n) = \sum_{k=0}^{N-1} d_k * e^{j2\Pi n\frac{k}{N}}, \qquad n = 0,1,2,\dots,(N-1)$$

where d_k is BPSK or QAM modulated signal, k is index of each subcarrier, n is a sample index, and N is a number of samples per symbol and also number of IFFT block points. Since above enclosed equation is actually IFFT, it is obvious that input signal in OFDM transmitter represents frequency domain samples. After IFFT block signal is divided in real and imaginary part. Each of them is now transformed from digital to analog signal, and then in mixer multiplied by signal which has frequency equal to frequency we want to transpose OFDM signal. Both signals are then summed and transmitted. Transmitted signal can be described by

$$x(t) = I(t)\cos(2\Pi f_c t) - Q(t)\sin(2\Pi f_c t)$$

where I is in-phase, and Q is quadrature component. I contains real part of OFDM symbol, and Q contains imaginary part of OFDM symbol before transposition. Also, as can be seen from figure 2.2. there is phase difference between signals which transpose OFDM baseband signal onto desired frequency.

In figure 2.3. is shown block scheme of OFDM receiver. Received signal in receiver is first multiplied by same signals as in transmitter in order to transpose it to baseband and to get real and imaginary part of OFDM signal. Received signal after multiplication with signal of certain frequency in first mixer (multiplied by $\cos(2\Pi f_c t)$) can be described by

$$x(t) = \frac{1}{2}I(t) + \frac{1}{2}I(t)\cos(2\Pi * 2f_c t)$$

and on output of other mixer, where signal is multiplied by $\sin(2\Pi f_c t)$, by

$$x_q(t) = \frac{1}{2}Q_k(t) - \frac{1}{2}Q_k(t)\cos(2\Pi * 2f_c t)$$

It is obvious that signal after multiplication contains one high-frequency component which is unwanted. This component is removed by low-pass filter. After that, filtered signal must be amplified by 2 and then transformed into digital signal by A/D convertor. After conversion, signal is again converted from serial to parallel stream and brought to input of FFT block and after Fast Fourier Transform is performed, signal is again converted into serial stream. Such signal is then demodulated and on demodulator output bit stream which contain information.

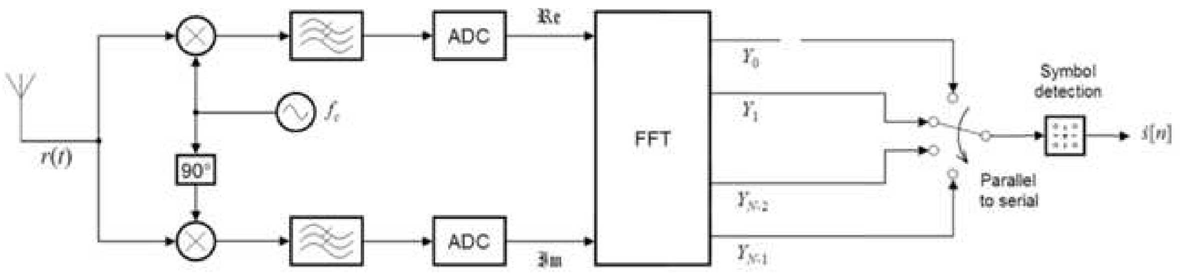

Figure 2.4. – OFDM receiver block scheme

During transmission of OFDM signal through the channel, signal can be affected by many effects. One of them is multipath propagation. The best explanation of multipath propagation is to say this is a channel with many paths (not just one such as in ideal condition). Each path does not have equal propagation time of signal through it. Some signal components will come in receiver before some other. Consequence of this effect will be Intersymbol Interference (ISI) because each OFDM symbol will be affected by preceding symbol. It is illustrated in figure 2.5. [6]. During the multipath propagation channel frequency response will not be ideal and so channel impulse response will not be ideal, as illustrated in figure 2.5. Symbol length will be enlarged by channel impulse response length and that will affect succeeding OFDM symbol and cause Intersymbol Interference and loss of information.

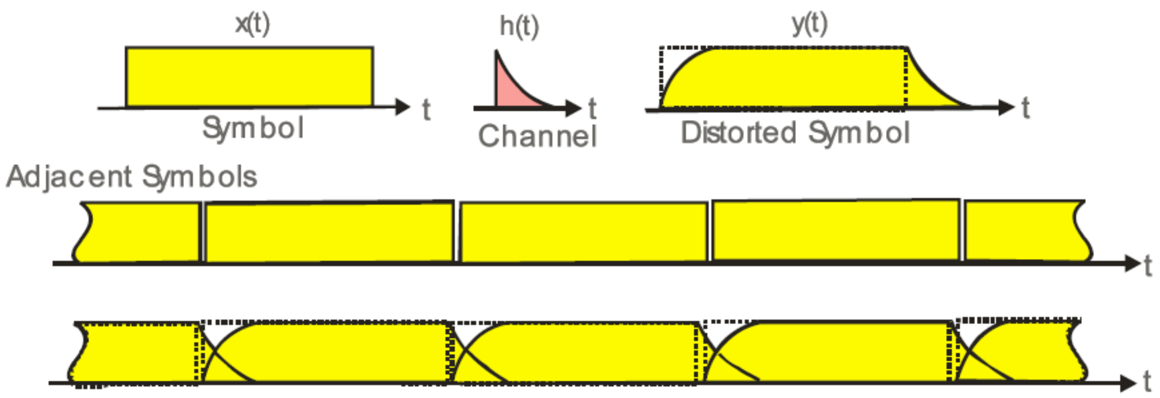

Figure 2.5. – Illustration of Intersymbol Interference

To avoid this, guard interval is introduced. Guard interval is a free space between OFDM symbols. That space is filled by last part of OFDM symbol and is called cyclic prefix. To avoid Intersymbol Interference, length of guard interval must be larger than propagation time in the channel. Otherwise, Intersymbol Interference will be present. Illustration of guard interval influence is shown on figure 2.6. [6].

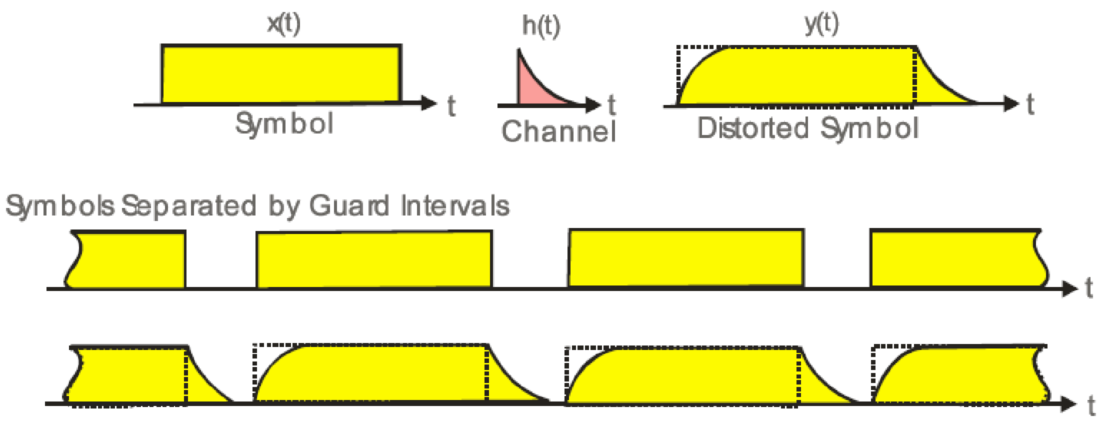

Figure 2.6. – Guard interval influence

2.2. MATLAB simulation model

Since it is necessary to perform certain simulations, MATLAB model is developed to this purpose. Communication channel is never ideal, and there will be performed some simulations. There are a lot of parameters in channel which can affect transmitted OFDM signal. In each simulation all parameter will be ideal, except one. In every paragraph problem will be presented, and then will be enclosed results of simulation to see how did each parameter affected OFDM signal.

This simulation model represents simulation of 802.11a standard for WLAN. For timing synchronization this standard utilizes short preamble which consists of four same symbols. Two short preambles are used, and also cyclic prefix to each preamble, so in total there is ten same symbols. To estimate frequency offset also short preamble is used. Long preamble is used for channel estimation. It consists of two symbols.

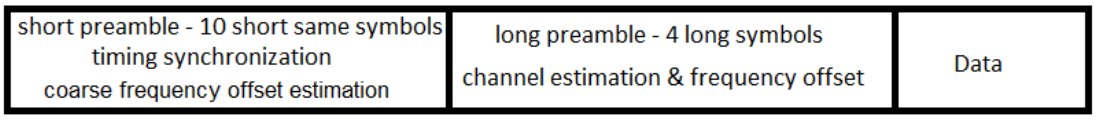
short preamble - 10 short same symbols
timing synchronization
coarse frequency offset estimation
long preamble - 4 long symbols
channel estimation & frequency offset
Data

Figure 2.7. – Preambles in 802.11a system

In MATLAB 802.11a model are not included all specifications of 802.11a standard. FEC and convolution coder are not included in system since the aim is just to check whether is received data accurate or not. Block scheme of system made in MATLAB is shown on figure 2.8. This system also represents complete OFDM system.

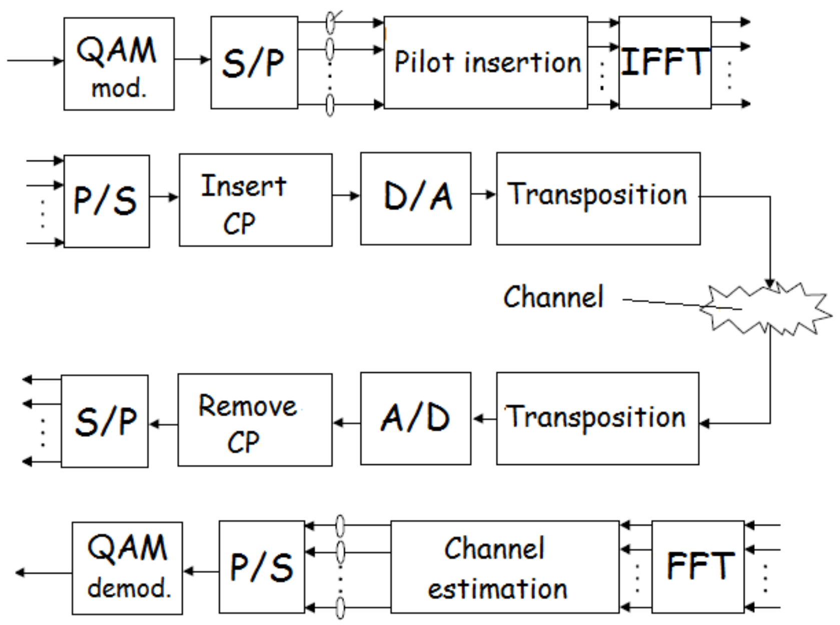

Figure 2.7. – Model of simulation system in MATLAB

In channel are introduced various disturbances, as will be explained in further text. First, we will assume that sampling frequencies in transmitter and receiver are same. Also, channel does not introduce any noise and has only one path (i.e. no multipath propagation). Parameter to be changed is frequency offset introduced in the channel. Depending on frequency offset there will be considered accuracy of timing metric. It will be illustrated by constellation diagram.

Constellation diagram in transmitter is shown in figure 2.9. It will be a reference for comparison. Same constellation diagram should be also in receiver in ideal conditions.

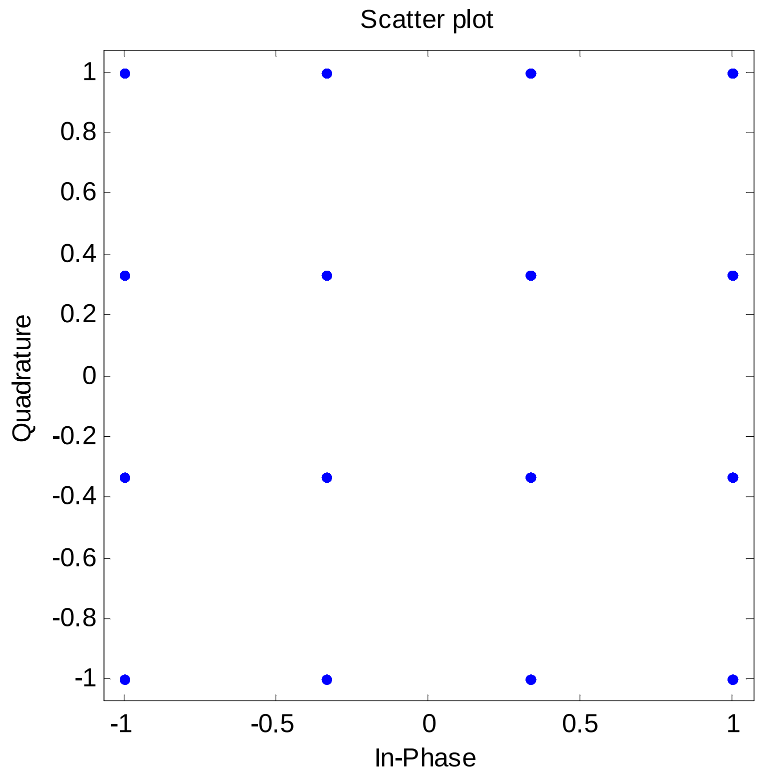

Figure 2.9. – Constellations in transmitter

As can be seen from the picture, 16-QAM modulation is used. In figure 2.10. and figure 2.11. is shown constellation diagram when there is present frequency offset in channel. Frequency offset is 0.01 rad in figure 2.10, and -0.01 rad in figure 2.11.

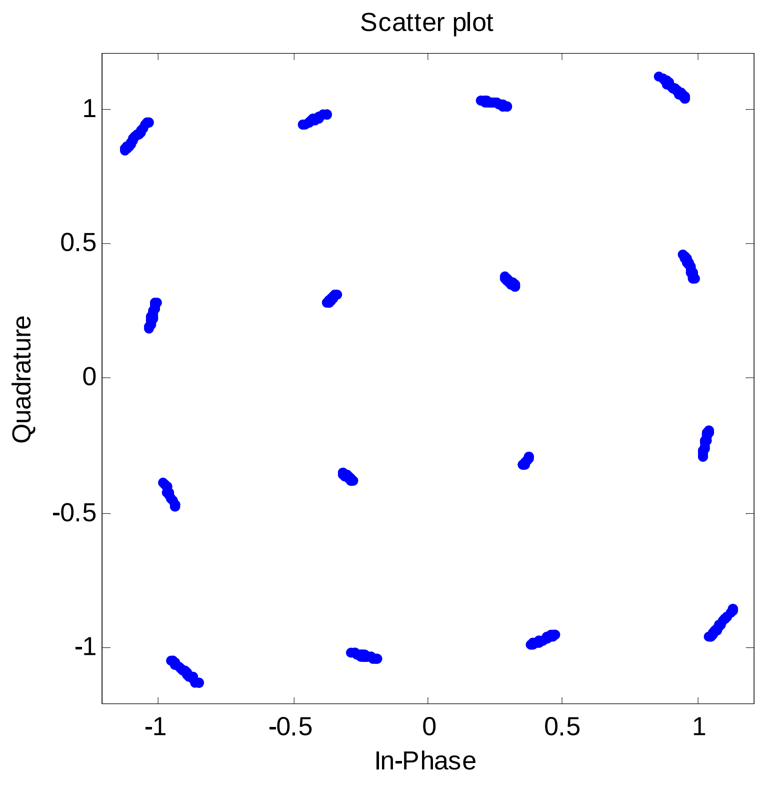

Figure 2.10. – Constellations in receiver for frequency offset of 0.01 rad

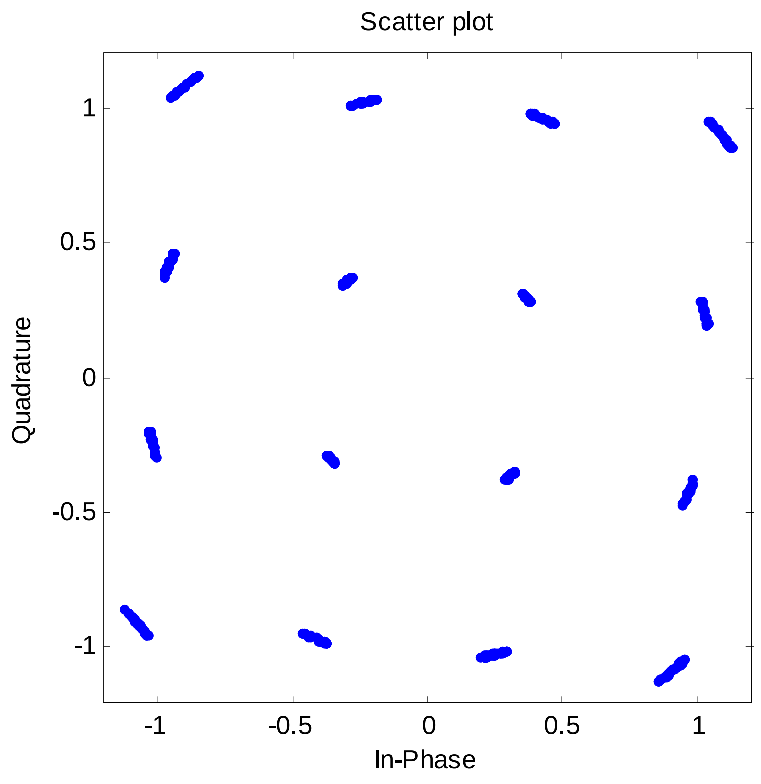

Figure 2.11. – Constellations in receiver for frequency offset of -0.01 rad

As can be seen on the picture, constellation diagram is not ideal any more. It is now rotated, either to left or right side, depending on sign of frequency offset value. But, there is still no error in transmission since the constellations are still within the bounds of each QAM symbol.

It is obvious that even a very little frequency offset can cause severe problems. To solve this problem, frequency offset estimator is used. Frequency offset is calculated as

$$\emptyset = \arg\big(P(d)\big)$$

Limitation of this estimator is that it can estimate accurately frequency offset just if it is within the interval [-π π]. If frequency offset is out this interval, constellation diagram in receiver will be as shown in figure 2.12.

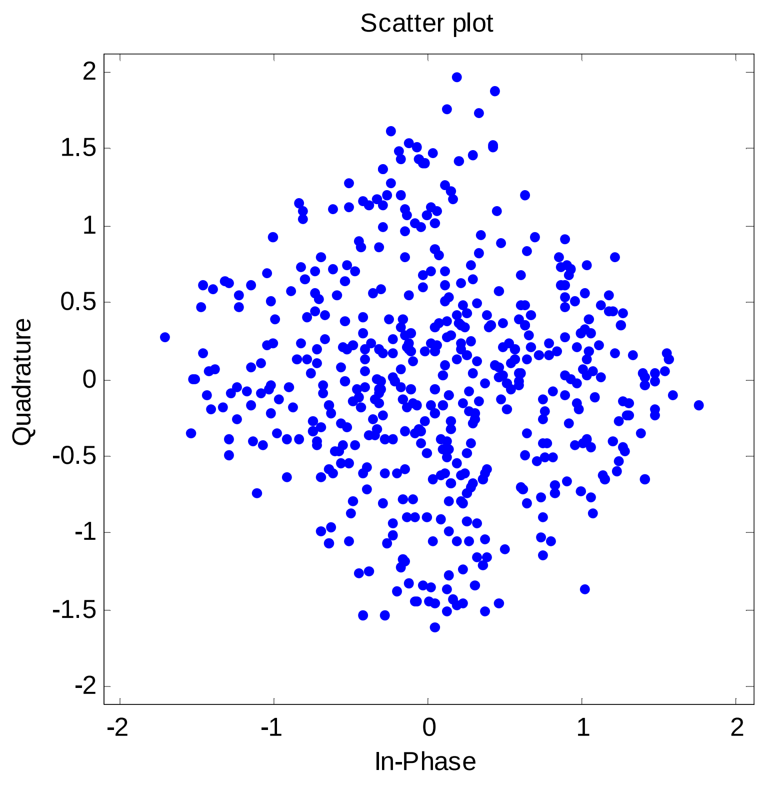

Figure 2.12. – Constellations for frequency offset of 3.2 rad

As expected considering theory, if absolute value frequency offset of each carrier is bigger than π, then there will be a huge bit error rate. For the case shown in figure 2.13., after transformation of QAM signals to bits, bit error rate is 0.5466, i.e. 54.66% of received bits are not valid.

To solve this problem, another algorithm is used. As mentioned earlier in text, total frequency offset can be represented by $v = 2 * q + u$ where q is an integer number which represents phase ambiguity and $u = \frac{1}{\pi} * \arg\big(P(\epsilon)\big)$ is a phase offset within [-π π] interval. Aim of algorithm is to estimate q, because then we will avoid phase ambiguity. Algorithm can be found in [3] (equation 18). Using this algorithm for same frequency offset as in figure 2.11., constellation diagram in receiver is shown in figure 2.12.

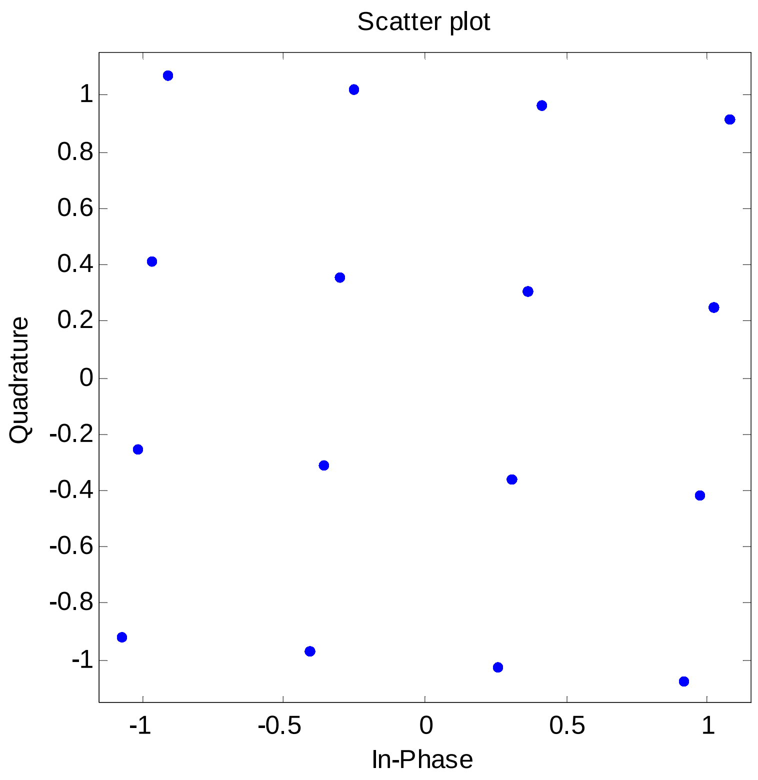

Figure 2.13. – Constellations in receiver during presence of frequency offset larger than π after compensation

At the end, it is good to say that beside the channel, LP filter in receiver also introduces some frequency offset since frequency response of filter is not ideal. This frequency offset is almost negligible.

After there were considered effects which are consequence of frequency offset, now there will be considered effects of sampling frequency offset. All other effects such as channel frequency response and carrier frequency offset introduced in the channel will be ideal there. If sampling frequencies in transmitter and receiver are same, constellation diagram in receiver will be same as in figure 2.8. As shown in figure 2.14., it can be seen that if sampling frequency in receiver is larger than sampling frequency in transmitter than each OFDM symbol window will contain data from succeeding OFDM symbol. Otherwise, if sampling frequency in receiver is lower than sampling frequency in transmitter, than one OFDM symbol window will not contain all information from the certain OFDM symbol.

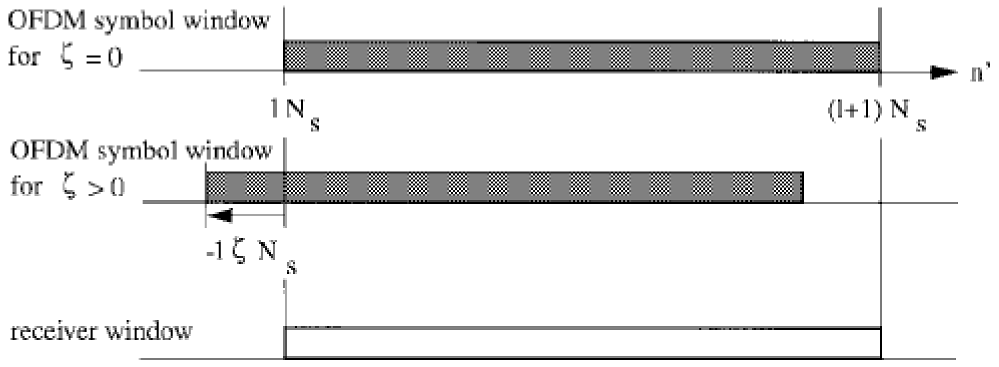

Figure 2.14. – OFDM symbol drift due sampling frequency offset

If sampling frequency in receiver is larger than in transmitter, constellations will be same as in figure 2.15.

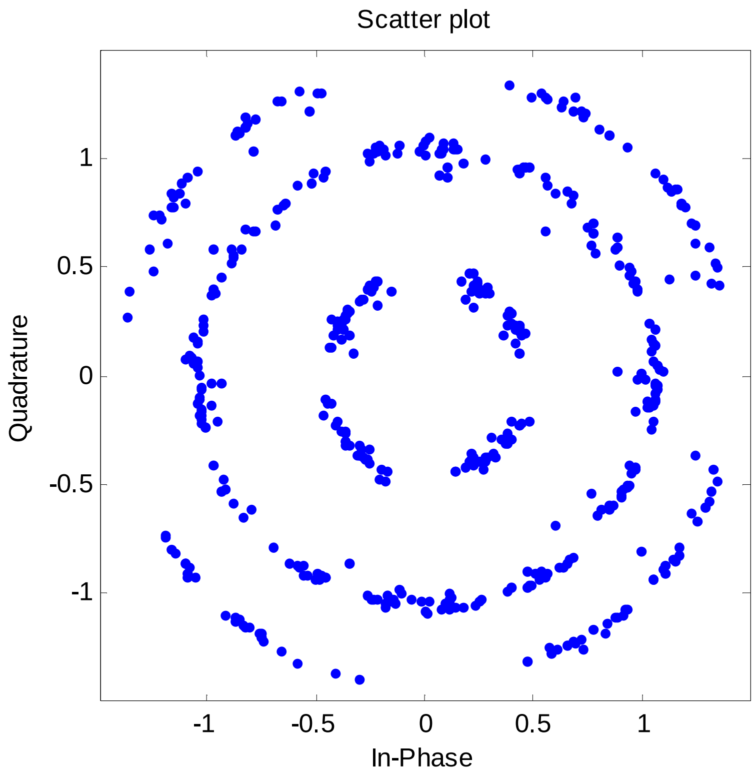

Figure 2.15. – Constellations during the sampling frequency offset

Sampling frequency in figure 2.15. is 20.004 MHz. Sampling frequency offset is represented by $\delta = (T' - T)/T$ where T is ideal sampling period and T' is sampling period in receiver. In this case, sampling frequency offset is 0.0002 (200 ppm) what is actually small value (Bit Error Rate is in this case 0.0135), but as can be seen, it causes a significant BER. During presence of sampling frequency offset problem is OFDM symbol drift [5]. Due to limitations in MATLAB and in hardware (lack of memory in laptop) it was not possible to simulate lower sampling frequency offset.

Simulation of Matlab model is performed to show effects of multipath propagation. This Matlab model is made according to 802.11a standard. In this standard duration of one OFDM symbol is 3.2 us and guard interval length is 0.8 us and sampling frequency is 20 MHz. Since it is necessary to make digital to analog conversion (D/A conversion) of signal, and in Matlab it is not possible, analog signal is represented by 50 times larger sampling frequency, i.e. it has 50 times samples more than digital signal in receiver. It must be considered during modeling multipath channel, because number of samples of guard interval in transmitter must be multiplied by 50, and that is maximum allowed length of multipath channel. In next figure is shown constellation diagram in receiver when multipath propagation is present. Length of channel is lower than length of guard interval.

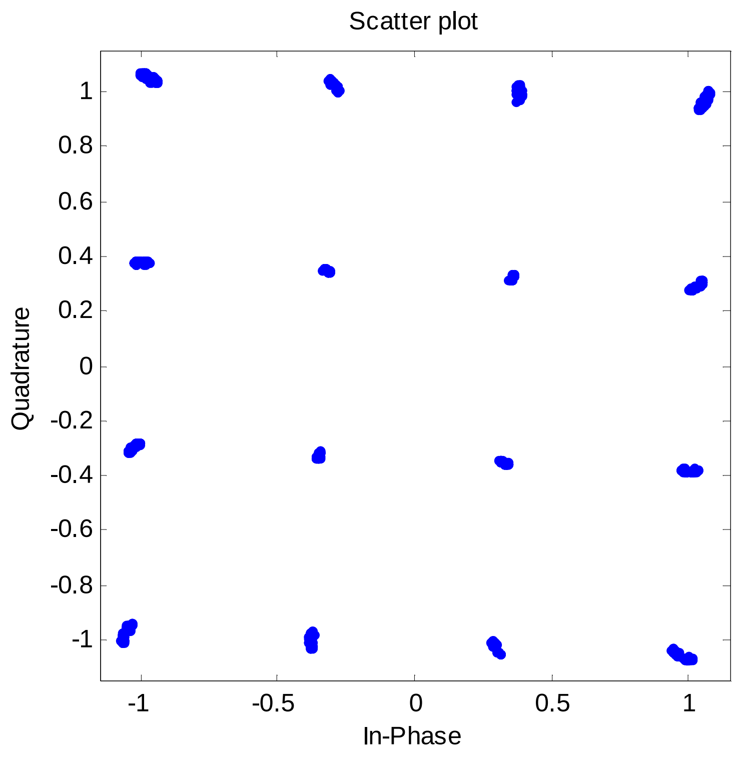

Figure 2.16. – Constellations in receiver during multipath propagation

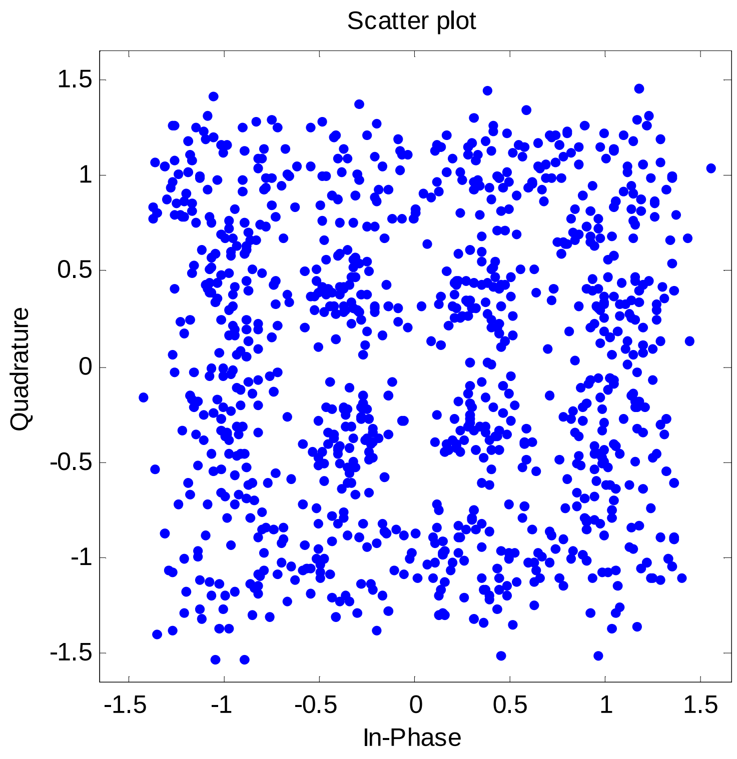

Figure 2.17. – Constellations in receiver (channel impulse response is longer than GIB)

In figure 2.18. is shown constellation diagram when channel impulse response is longer than guard interval. It is obvious that BER is present since certain constellations are not within their bounds. As guard interval is longer, robustness to multipath propagation problems is increased, but data rate will suffer, i.e. it will be lower.

After this simulations now will be performed some other simulations. First, there will be performed simulation where will be considered Bit Error Rate depending on signal to noise radio in the channel. All other parameters which were changed in earlier simulations will be ideal in this case. According to figure 2.17., it is obvious that for ratios under the 30 dB Bit Error Rate is present.

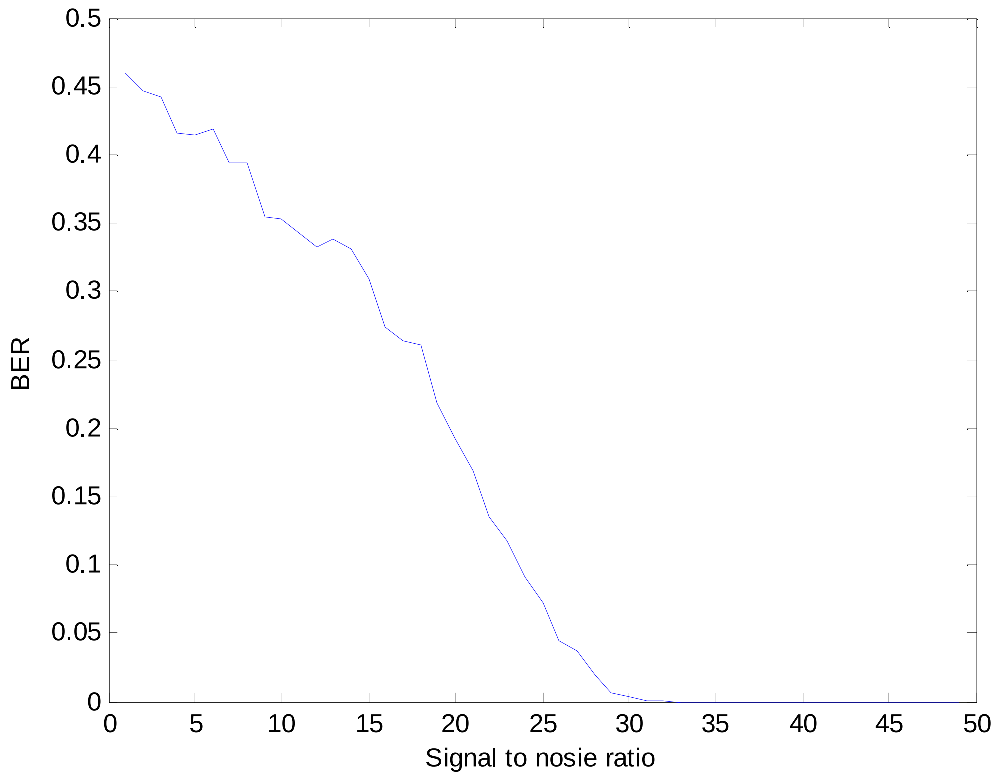

Figure 2.18. – Bit Error Rate depending on signal to noise ratio

Since channel is not ideal and there is always added noise to signal during the transmission which changes channel frequency response. It is possible to estimate channel frequency response using long preamble and then compensate received signal. It is possible to estimate channel frequency response just on the frequencies of carriers and that is actually only what is needed. Channel estimation can be described by

$$H(k) = \frac{Y(k)}{X(k)}$$

where $H(k)$ is estimated channel frequency response, $Y(k)$ is received data after Fast Fourier Transform is performed, $X(k)$ is sent data, and k is index of each subcarrier. Sent data $X(k)$ is defined by standard and is known in receiver. In case of 802.11a standard it is long preamble. After channel is estimated, to compensate signal, each received OFDM symbol is divided by $H(k)$. Compensation of signal is never ideal, but can significantly repair received signal affected by various disturbances in the channel.

3. **Synchronization in OFDM systems**

When receiver detect OFDM (and any other signal) first step to do is to determine start of the frame which contains two or more OFDM symbols. There is a certain number of methods for but there will be analyzed and discussed two of them. One method is based on preamble, which is actually one OFDM symbol divided in two identical halves. Other method is cyclic prefix based (also, often denoted as GIB – Guard Interval Based) and it uses cyclic prefix to detect start of the frame. Also, those methods at the same time perform a coarse frequency offset estimation. This also will be analyzed in the further text. Since Guard Interval Based method is designed and used in broadcast systems, there will be introduced some modifications to be able to use this method for other wireless systems which are not broadcasting systems.

For better understanding, there will be enclosed and explained some equations which represents effects we are analyzing here. OFDM symbol can be described by

$$s(n) = \sum_{k=0}^{N-1} d_k * e^{j2\Pi n\frac{k}{N}}, \qquad n = 0,1,2,\dots,(N-1)$$

where d_k is BPSK or QAM modulated signal, k is index of each subcarrier, n is a sample index, and N is a number of samples per symbol and also number of FFT block points. If signal is affected by carrier frequency offset, then signal can be represented in receiver by

$$r(n) = \mathrm{s}(n) * e^{2\Pi n\frac{v}{N}}$$

where v is a carrier frequency offset. If we also introduce and timing offset then signal will be represented by

$$r(n) = \mathrm{y}(n-m) * e^{2\Pi n\frac{v}{N}}$$

where m is unknown integer delay of OFDM signal. y is convolution result of channel impulse response and original signal, so it can be represented by

$$y(n) = \sum_{k=0}^{L-1} h(k) * s(n-k)$$

where h is a channel impulse response, and L is length of channel. Because received signal must be almost same as a sent signal to get an accurate data, it is necessary to apply some methods and algorithms in order to repair received signal. Two above mentioned methods and their variants will be described in next two chapters. In further chapters they will be analyzed in terms of computational complexity and implementation to use as less as possible resources of available platform. Also, there will be considered and other properties. Probability of errors of each method, variances and mean square errors of each method depending on signal to noise ratio introduced in communication channel. Since idea is to analyze methods based on two different basic principles, for analysis will be chosen best versions of each method.

Preamble based method	**GIB based method**
Schmidl and Cox	Moving Average
Ren Method	Exponentially Moving Average
Modified Schmidl and Cox	
Sliding Window Method	
Autocorrelation Method	
Correlation Based Estimator	
Double Autocorrelation Method	

Table 1 – List of methods

3.1. Preamble based synchronization method

To detect start of the frame, this method uses a preamble. Preamble is, as mentioned before, OFDM symbol divided in two halves containing identical data. In order to determine start of the frame, correlation of these two halves is performed. The aim is to detect a peak in correlation result, i.e. to find the highest value in result. Since there are many versions of this method some of them will be briefly described in this chapter and advantages and disadvantages will be considered.

3.1.1. Schmidl and Cox method

Probably the most simple method is Schmidl and Cox method [1]. This method uses two OFDM symbols with two identical halves. Each of them contains a PN sequence. In first symbol, data is contained just on even indexed carriers, and on odd indexed carriers are just zeros in order to reduce large amplitude changes. Halves will be identical in receiver, but there will be a phase shift caused by frequency offset. Second symbol contains two PN sequences, one on even, and one on odd indexed subcarriers in order to determine frequency offset. For timing synchronization, correlation of first symbol halves is performed. According to [1] it can be represented by

$$P(d) = \sum_{i=0}^{L-1} r^*(d+i) * r(d+i+L)$$

where r is received symbol and L is half of length of a OFDM symbol. Energy of second half of symbol is defined by

$$R(d) = \sum_{i=0}^{L-1} r(d+i+L)^2$$

Finally, timing metric is determined by

$$M(d) = \frac{P(d)^2}{R(d)^2}$$

Start of the symbol corresponds to index d for which the value of M is maximum. Because of cyclic prefix presence while performing correlation, result of timing metric will contain a plateau and it leads to ambiguity since there will be probably more than one maximum value or difference between them will be quite small.

To estimate carrier frequency offset both preambles are used. Since in simulation model in MATLAB developed for this purpose are two preambles which are not suitable for this method, there will be introduced some changes in preambles according to [1]. There should be zeros on odd subcarriers and PN sequence on even subcarriers of first preamble and this is not a case in first (short) preamble defined by 802.11a standard. Also, there is necessary to make some changes in second (long) preamble since there are no complex numbers as it should be according to [1]. Long preamble is used for channel estimation, but since it is not necessary to make here it will not cause any problems. Frequency offset can be estimated as

$$\emptyset = \arg\bigl(P(d)\bigr).$$

If calculated phase difference ∅ is between –π and π frequency offset can be calculated by $\Delta f = \emptyset/\pi\mathrm{T}$. When received signal is multiplied by this offset, adjacent carrier interference is avoided. But there still could be present other frequency offset, and is equal to $\Delta f = 2\mathrm{z}/\mathrm{T}$, where z is an integer. To calculate remaining frequency offset FFT must be performed. Let $x_{1,k}$ be a FFT transform of first preamble, $x_{2,k}$ FFT of second preamble, v_k their ratio, and $X = \{-M, -M + 2, \ldots - 2, 2, \ldots, M - 2, M\}$ indexes of even subcarriers. Integer number z is determined by maximum value of $B(z)$ which is calculated as

$$B(z) = \frac{|\sum_{k \epsilon X} x_{1,k+2z}^{*} v_k^{*} x_{2,k+2z}|^2}{2(\sum_{k \epsilon X} |x_{2,k}|^2)^2}$$

3.1.2. Ren Method

Carrier frequency offset is calculated by $u = \frac{1}{\pi} * \arg(P(\epsilon))$. ϵ is sample which corresponds to correct starting sample. Since phase has a values between –π and π (or 0 to 2π), range of this estimation is limited on interval -1 to 1. If frequency offset is bigger, it can be represented by $v = 2 * q + u$ where q is an integer number which represents phase ambiguity. Algorithm which calculates this number will be given in further text. Let $r(n)$ be a received signal which has certain frequency offset. This signal is multiplied by $e^{-j2\pi vn/N}$ and the result is $r_1(n)$. Received signal is compensated by v. To compensate remaining frequency offset $2 * q$, signal $r_1(n)$ must be multiplied by unaffected preamble (multiplied by PN sequence), as $r_2(n) = r_1(n) * p(n)$, where $p(n)$ is original preamble. q is defined by maximum value of which is defined as

$$I(q) = |\sum_{n=0}^{N-1} r_2(n) e^{-j*4\pi qn/N}|^2$$

where $q = \{-\frac{N}{4}, \dots, -1, 0, 1, \dots, \frac{N}{4}\}$.

3.1.3. Sliding Window Method

There are two modifications of Schmidl and Cox method. According to [2] first one is called *Sliding Window Method*. First modification is that half of signal energy is calculated over all samples instead of just last part of OFDM symbol.

$$R(d) = 0.5 * \sum_{i=0}^{N-1} r(d+i)^2$$

where N is number of samples in OFDM symbol. Second, and more important modification is averaging the timing metric. Second change is averaging timing metric over last N_g samples what is a number of samples of cyclic prefix.

Timing metric $M(d)$ calculated in the same way as in previous methods is averaged as

$$M_1(d) = \frac{1}{N_g + 1} * \sum_{i=-N_g}^{0} M(d+i)$$

3.1.4. Modified Schmidl and Cox Method

This method is actually modified Schmidl and Cox method. In Schmidl and Cox method correlation is performed over two symbols of the short preamble, i.e. over 128 samples, since each symbol is 64 samples long. Because of presence of cyclic prefix, which is same as last part of the symbol, there will be a plateau in timing metric which can lead to ambiguity and decrease accuracy of timing metric. To avoid plateau, correlation will be performed at same time over all 160 samples of short preamble, including cyclic prefix. Each of two symbols has four equal parts, each of them 16 samples long. Cyclic prefix is also 16 samples long. Correlation is calculated as

$$P(d) = \sum_{i=0}^{L+G-1} r^*(d+i) * r(d+i+L+G)$$

where L is length of symbol, and G is length of guard interval. Energy is calculated as

$$R(d) = 0.5 * \sum_{i=0}^{2(L+G)-1} r(d+i)^2$$

Timing metric is calculated as ratio of these two results, as

$$M(d) = \frac{P(d)^2}{R(d)^2}$$

Since in 802.11a system it is not possible to apply this method, some changes are introduced.

3.1.5. Double Autocorrelation Method

This method in order to detect start of the frame calculates two correlations, according to

$$P_1(d) = \sum_{i=0}^{N_S-1} r^*(d+i) * r(d+i+N_s)$$

$$P_2(d) = \sum_{i=0}^{N_S-1} r^*(d+i) * r(d+i+2N_s)$$

where N_s is a length of each short symbol. Energy is calculated as

$$E(d) = \sum_{i=0}^{N_S-1} |r^*(d+i)|^2$$

Each metric is calculated as

$$M_1(d) = \frac{P_1(d)^2}{E(d)^2}$$

$$M_2(d) = \frac{P_2(d)^2}{E(d)^2}$$

Finally, timing metric is calculated as

$$M = argmax(M_1 - M_2)$$

In ideal conditions, this method will detect start of the 9th short symbol. Start of the 9th short symbol corresponds to maximum value of result after subtracting two metrics.

3.1.6. Correlation Based Estimator

This method does not include a timing metric, so it will be only considered for frequency offset estimation. Frequency offset in this method is calculated in frequency domain, so it is necessary to perform FFT. Let P be preamble in frequency domain. After FFT is performed, correlation in frequency domain is calculated as

$$R(\epsilon) = \sum_{n=-N/2}^{N/2-1} P_n^* * P_{n-\epsilon}$$

where ϵ is integer frequency offset and is determined by maximum value of $R(\epsilon)$. Fractional frequency offset is calculated as $\emptyset = \arg\big(P(d)\big)$ where $P(d)$ is correlation of preamble, and $\emptyset$ is fractional frequency offset multiplied by π.

3.1.7. Autocorrelation method

This method is used just for coarse estimation of frequency offset. Correlation is calculated as

$$K = \sum_{k=0}^{15} r^*(k) * r(k+16)$$

Let $|\mathrm{K}|$ be

$$|K| = \sum_{k=0}^{15} |r(k+16)|^2$$

Frequency offset is calculated as

$$\varepsilon = -\frac{2}{\pi}\arg\left(\frac{K}{|K|}\right)$$

3.2. GIB based method

As stated before, this method for timing and carrier frequency offset estimation uses a redundant part of OFDM symbol called guard interval (cyclic prefix). Similarly as in preamble based method, aim is to detect a peak in result of correlation. In [4] are presented some versions of timing synchronization. One is called MA (*Moving Average*) and is represented by

$$P(d) = \sum_{i=d-L+1}^{d} Re\{c^*(i) * c(i-N)\}$$

in time instant d. c is quantized received signal, N is length of OFDM symbol and L is length of cyclic prefix. Second method is *Exponentially Moving Average* and is represented by

$$P(d) = \sum_{i=0}^{d} \mathrm{w}^{d-k} Re\{c^*(i) * c(i-N)\}$$

w is a weight factor.

Algorithm which calculates a carrier frequency offset is presented in [4]. It will be enclosed in further text. Carrier frequency offset and timing estimation are not independent. Let frequency offset be ϵ. It is calculated as

$$\epsilon = tan^{-1}\frac{\sum_{k=\theta-L+1}^{\theta} Im\{c(k) * c^*(k-N)\}}{\sum_{k=\theta-L+1}^{\theta} Re\{c(k) * c^*(k-N)\}}$$

4. Simulations

In this chapter will be presented results of simulations for each method. As it is mentioned earlier in the text, each method is applied to the 802.11a standard. Specifications of 802.11a standard can be found in [7]. There are in first subchapter analyzed methods to estimate coarse frequency offset and in second one methods to estimate start of the frame. Every method is analyzed in terms of mean squared error, variance and for timing synchronization also in terms of probability of accurate detection.

4.1. Frequency offset methods

There are two types of simulations performed for frequency offset methods. First one is to see the estimation range for each method, and the second one is to measure accuracy of each method by calculating mean squared error and variance. During the measurement of mean squared error and variance, it was assumed that start sample is already known because in this measurement it was not aim to measure timing synchronization. Also, it was the aim to measure mean squared error and variance considering signal to noise ratio. Since for signal to noise ratio for above the 30 dB, measurement was performed in range of 0 to 30 dB. Methods analyzed for frequency offset are shown in table 2:

Method	**Range**
Schmidl and Cox method	-π to π
Autocorrelation method	-15.75 to 15.75
Correlation Based Estimator	-π to π
GIB based method	-π to π
Ren Method	-20 to 20

Table 2 – Methods used for coarse frequency offset estimation

As described in previous chapter, Schmidl and Cox method and correlation based method are methods just to estimate frequency offset if it is larger than π. They are calculate frequency offset after Fast Fourier Transform is performed. To be able to estimate frequency offset those methods need to have special content of preamble. As shown in this chapter after performed simulations it is obvious that those methods can not calculate frequency offset larger than π using preamble in 802.11a system.

Range of other three methods is shown at figure 4.1.

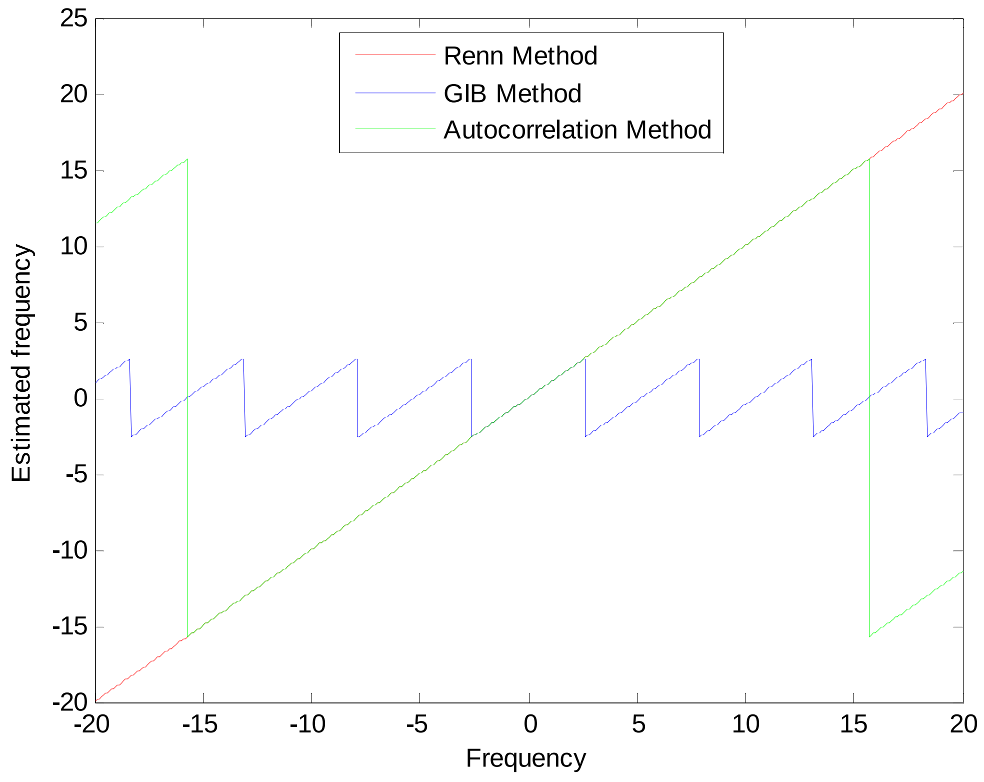

Figure 4.1. – Estimation range for each method

At figure 4.1. it can be seen that Renn method has ability to estimate largest frequency offsets. Frequencies are not normalized, there are shown real frequencies in radians. Range of each frequency is measured on range -20 to 20 radians.

Accuracy of each method is determined by mean squared error and variance. Mean squared error is shown at figure 4.2.

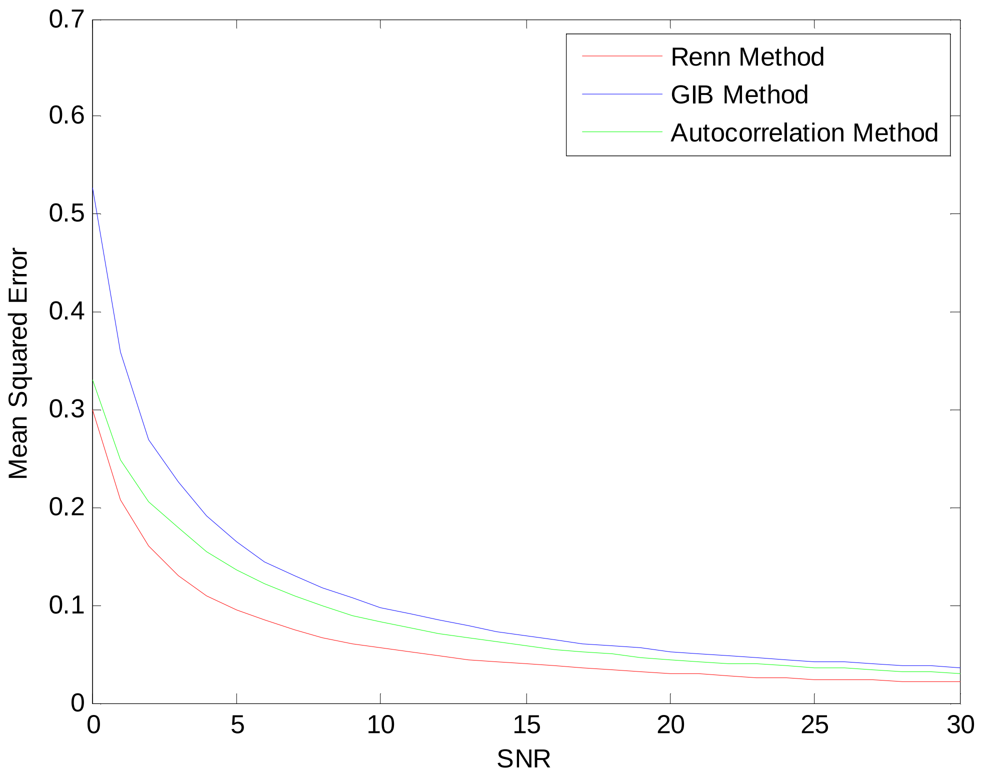

Figure 4.2. – Mean squared error of each method

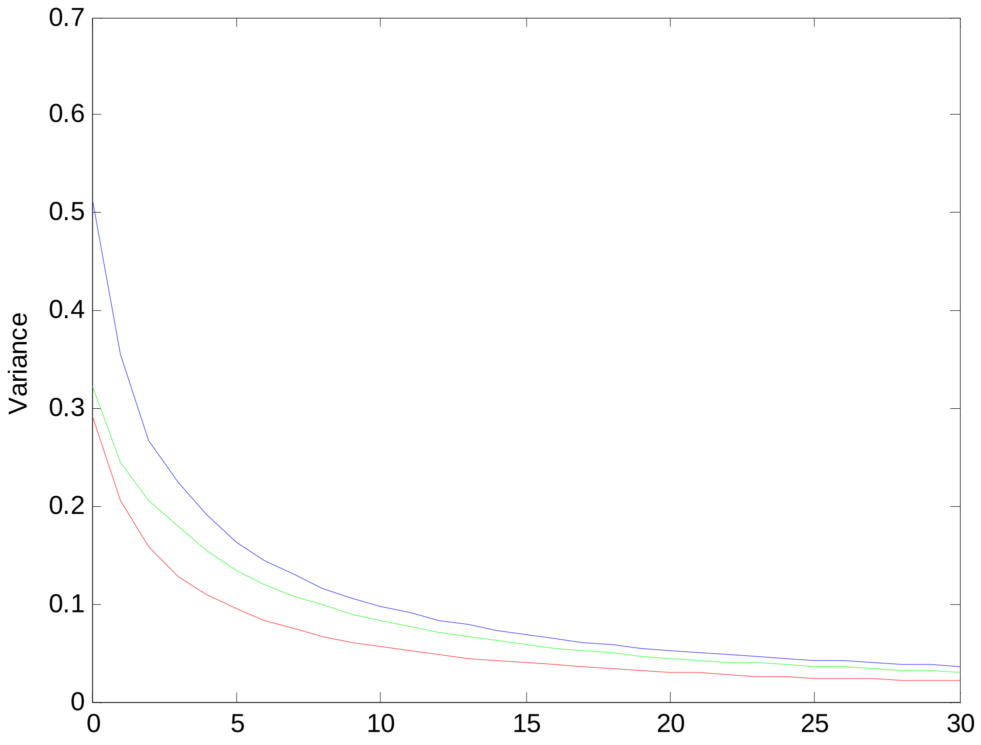

Figure 4.3. – Variance of each method

According to these figures, it is obvious that Renn Method has the best performances in terms of mean squared error and variance. It means that this method is actually the most accurate, i.e. it has least deviation from real frequency.

4.2. Timing synchronization methods

In simulations of timing synchronization methods there were performed three types of simulation. First one is measuring mean squared error, second one variance, and third one determines probability of accurate detection of start of the frame. As in frequency offset methods, mean squared error, variance and probability of accurate detection performances were measured considering signal to noise ratio. Range of signal to noise ratio was from 0 to 40 dB. Above 40 dB there are negligible differences. Mean squared error of each method is shown in figure 4.4.

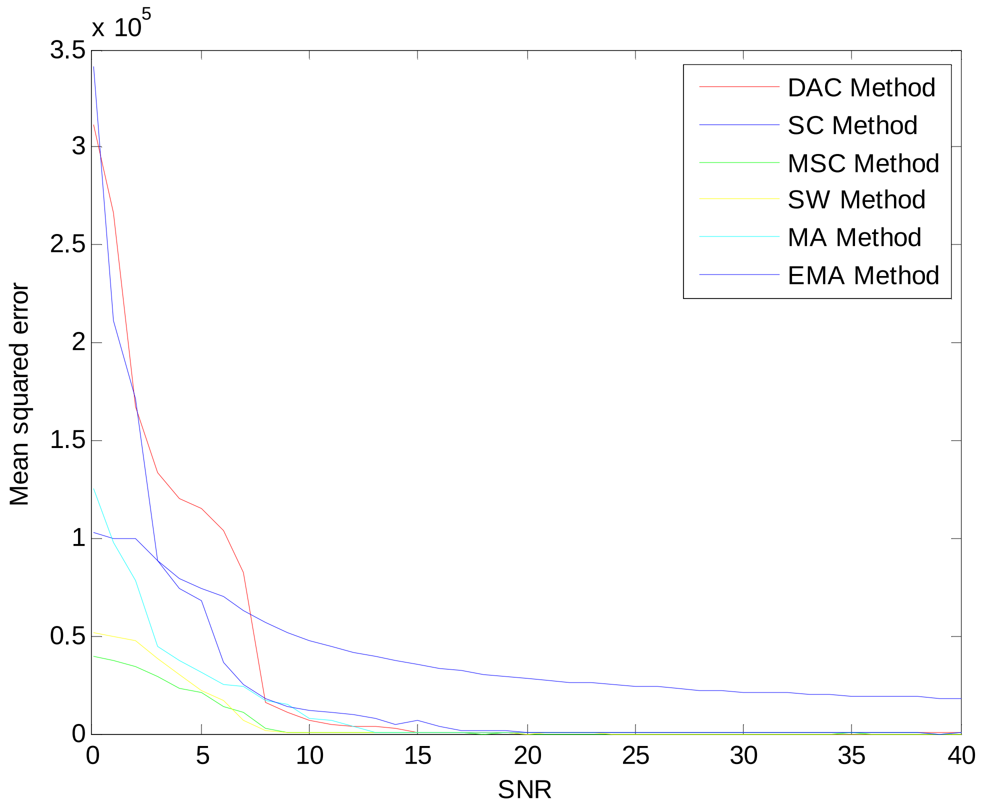

Figure 4.4 – Mean squared error

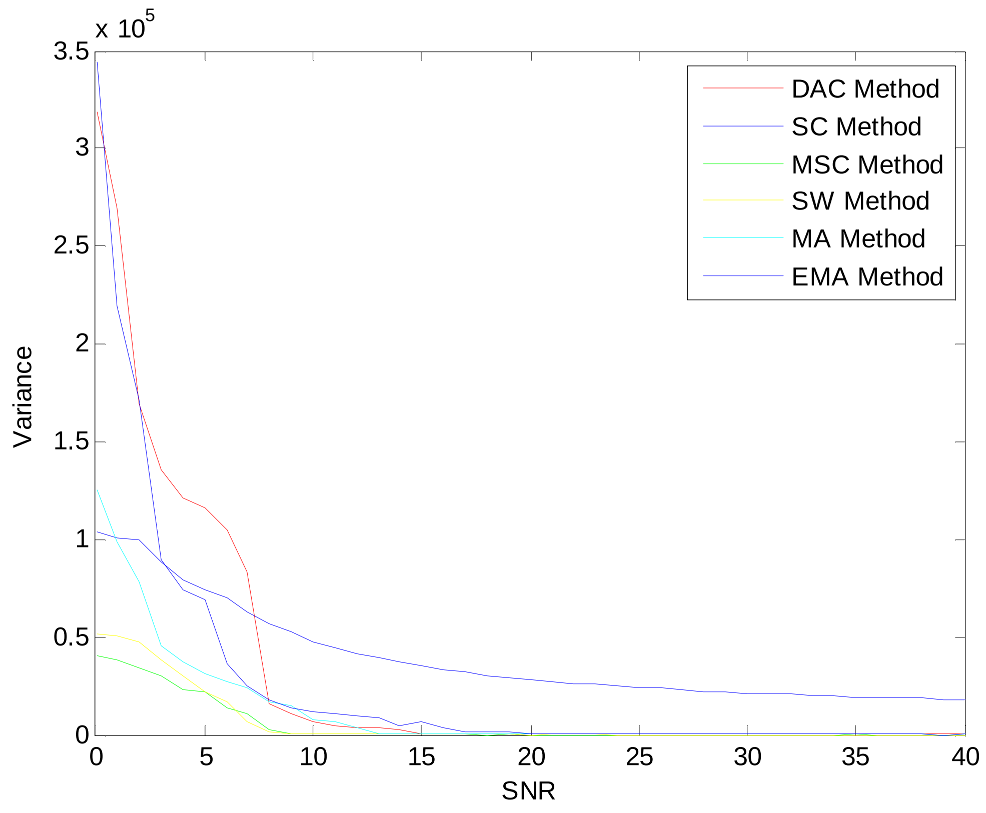

Figure 4.5. – Variance of each method

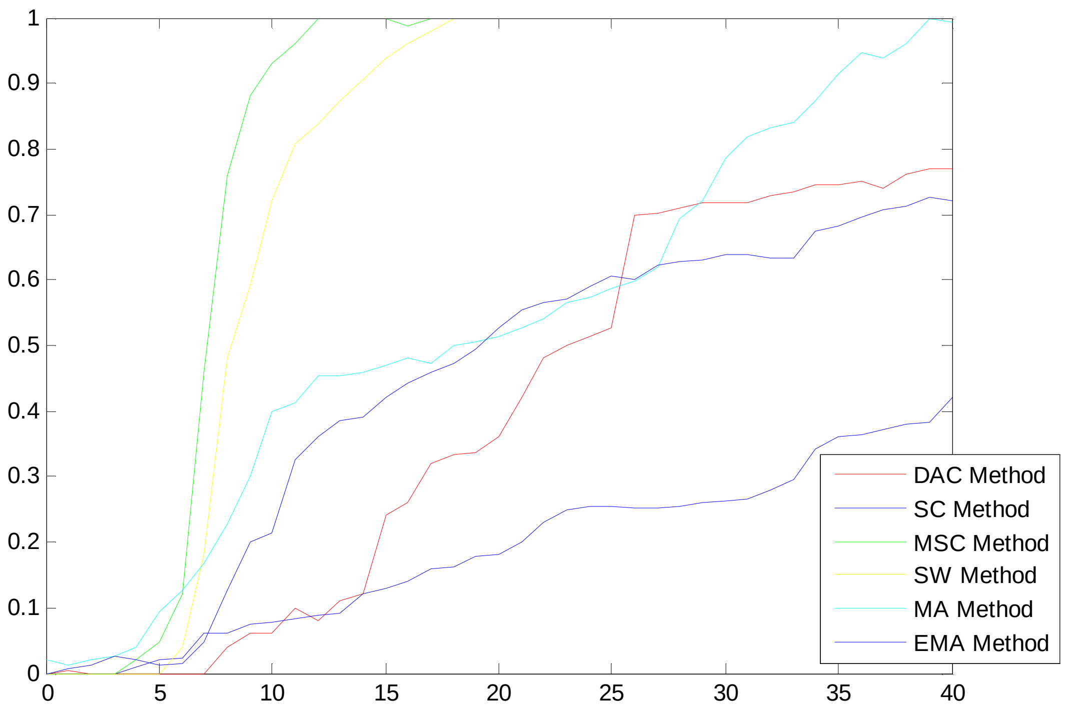

Figure 4.6. – Probability of accurate detection of each method

According to measurements shown on figures 4.4 and 4.5. the smallest mean squared error and variance has Modified Schmidt and Cox method. Also, according to performed measurements shown in figure 4.6., this method has the largest probability of accurate detection.

5. Architectural design

This chapter deals with implementation of one part of receiver which performs timing synchronization and coarse estimation frequency offset. As it is mentioned before, according to results of measurements, two methods chosen for implementation are Renn Method for coarse frequency offset and Modified Schmidt and Cox method. In further text there will be more about platform chosen for implementation. To implement whole system, there will be derived signal data flow graph and precedence graph. After that, next step is scheduling, i.e. time is introduced into precedence graph (defining which operation is performed in each clock cycle).

5.1. Platform for implementation

There are several options for implementation of this part of 802.11a receiver. ASIC (Application Specific Integrated Circuits) are integrated circuits which performs some specific task. They can contain elements such as multipliers, adders, registers, etc. These integrated circuits are fixed-wired and they are not reconfigurable. It is suitable for mass production, because huge amounts of money must be invested in devices which produce them.

Other very popular platforms for realization of DSP algorithms are DSP processors and FPGA (Field Programmable Gate Array). Digital Signal Processor is actually special type of microprocessor which is adapted to perform repeatable mathematical operations. Speed of DSP processor is limited by its own frequency (clock rate). Also, other limiting factor is number of certain elements, such as adders and multipliers because if processor contains just one adder, than in each clock cycle there can be performed only one multiplication, but if there are more multipliers (e.g. 4) then in one clock cycle there can be four multiplications. Digital Signal Processors are usually programmed in C language, but they can also be programmed in assembler because then you have bigger control on processor resources.

FPGA, as its name says, is a field of programmable gates. It consists many programmable components called logic blocks. They contain many logic gates such as AND, OR and XOR, distributed RAM memories which can be used as memory or for some logic function. Other significant parts that FPGA contains are multipliers, block RAM memories, configuration logic, and wiring. FPGA is programmed by connecting each logic gates in logic blocks, and logic blocks with multipliers and block RAM. It is usually programmed in some hardware description language, and probably most popular such language is VHDL (Very high speed integrated circuits

Hardware Description Language). FPGA platform is reconfigurable, as well as DSP processors.

It is obvious that ASIC is not suitable for implementation in this project because there is going to be made just one device, and it is not for mass production. Other two options are DSP processors and FPGA and there are many elements which determine which one should be chosen for implementation. If there is high sampling rate required (above a few MHz) then FPGA is a lot better choice. It contains more adders and multipliers and can perform more operations simultaneously. This system has according to 802.11a standard sampling rate of 20 MHz what is relatively high sampling rate, so FPGA is in this terms much better choice than DSP. Also, there are not a lot of conditional operations and that is one more reason to choose FPGA. DSP has advantage if there is used floating point arithmetic but it is not case here. Everything will be implemented in fixed-point arithmetic.

There are several FPGA development kits (development boards). Which will be chosen, it will depend on analysis in further text.

5.2. Algorithmic specifications

As already is known, algorithm to be implemented is

$$P(d) = \sum_{i=0}^{L+G-1} r^*(d+i) * r(d+i+L+G)$$

$$R(d) = 0.5 * \sum_{i=0}^{2(L+G)-1} r(d+i)^2$$

$$M(d) = \frac{P(d)^2}{R(d)^2}$$

for timing synchronization, and for coarse frequency offset

$$I(q) = |\sum_{n=0}^{N-1} r_2(n) e^{-j*4\pi qn/N}|^2$$

$$\emptyset = \arg\big(P(d)\big).$$

To be able to start to calculate coarse frequency offset, start sample must be determined already. It is calculated over 160 samples and that means that 160 samples must be stored in buffer or memory. Since all time new samples are going in, memory should be organized as FIFO (First In First Out). Also, it must have two separated places, one for real, and one for imaginary values since samples are complex numbers.

To multiply those numbers there must be performed four multiplications. In general, we can write first complex number as

$$r_1 = x_1 + jy_1$$

and second one as

$$r_2 = x_2 + jy_2$$

Expression for multiplication can be written also as

$$P = x_1 * x_2 + jx_1y_2 + jx_2y_1 - y_1y_2$$

There are four multiplications and three additions to be performed to multiply two complex numbers. Between two consecutive samples, there need to be executed multiplications of 80 complex numbers. Also, there must be executed 79 additions to calculate correlation. Since per one multiplication of complex numbers there must be executed four multiplications and three additions, that means that between two consecutive samples must be executed 320 multiplications and 319 additions and it is really too many operations just for correlation. To reduce it, there will be introduced some changes in the way of calculating correlation. Two consecutive correlations, $P(d)$ and $P(d+1)$ there are 159 same samples used for its calculation. To reduce complexity of calculation, to calculate new correlation result of previous correlation will be used. We can denote $P(d)$ for old correlation result (the one already calculated) and $P(d+1)$ for new correlation result. First one is calculated as

$$P(d) = r^*(d) * r(d+L) + r^*(d+1) * r(d+1+L) + \cdots + r^*(d+L-1) * r(d+2L-1)$$

Second (new) correlation can be calculated as

$$P(d+1) = r^*(d+1) * r(d+L+1) + r^*(d+2) * r(d+L+2) + \cdots + r^*(d+L) * r(d+2L)$$

It is obvious that new correlation can be calculated executing just two multiplication of complex numbers and two additions instead of 80 multiplications and 79 additions. If old intermediate results are stored in other buffer, it can be reduced to just one multiplication of complex numbers and two additions because there is not necessary to execute one more multiplication of complex numbers (to multiply samples $r(d)$ and $r(d+L)$ which has already been calculated before). In total, it is in first case eight multiplications and additions, and in second case, when buffer is used to store old results, it is just four multiplications and five additions. Since sampling rate of system is relatively large, and FPGA contains enough memory resources, what is not case with multipliers, second option will be chosen.

Also, there must be calculated an energy of a signal. There is also necessary to have 160 samples to calculate this and there is also necessary to calculate 79 additions and 80 multiplications of complex numbers. This multiplication is just squaring, multiplying complex number with itself. To reduce complexity, same principle will be applied as for calculation of correlation. As mentioned before, since FPGA has enough memory resources, another 160 intermediate results will be stored in certain buffer. After correlation and energy are calculated, next to calculate is normalized correlation. To calculate this, there is only need to execute one division. Since there is no dedicated divider in FPGA, one must be made. That what is mentioned in this paragraph would significantly increase computational complexity of device.

Finally, after all above mentioned operations are executed, there is just to determine starting point of the frame. Threshold is equal to 0.5, so any result of normalized calculation lower than threshold is not important. After result of normalized correlation bigger than 0.5 is detected, it will be stored in buffer. If new result of normalized correlation is bigger than 0.5 and than previous result, then this result will be stored in the buffer. This process will be repeated until result of correlation is smaller than 0.5. The biggest among all results bigger than 0.5 is the starting sample. When correlation with value bigger than 0.5 is detected, even more samples than 160 must be stored. That is because after the biggest result (starting sample) there still can be more results with value bigger than 0.5 and it is necessary to check. This is relatively small number of samples above 160 which need to be stored (memory size will be specified later).

Second part of receiver calculates coarse frequency offset after start of the sample is already found. First there must be calculated angle. It can be calculated as arctangent function. To implement this, there are two main opportunities. One is CORDIC algorithm and second one is look-up table.

CORDIC (Computer Rotation Digital Computer) algorithm is iterative algorithm which adds certain angle to starting vector. For example, if we want to compute arctangent of complex number $r_0 = x_0 + jy_0$ it is calculated as

$$x_{i+1} = x_i - y_i d_i * 2^{-i}$$

$$y_{i+1} = y_i + x_i d_i * 2^{-i}$$

$$z_{i+1} = z_i - d_i * \arctan(2^{-i})$$

where $d_i = \begin{cases} 1, y_i < 0 \\ -1, y_i \geq 0 \end{cases}$. i is number of iterations and by i accuracy of result is determined. Results of operation $\arctan(2^{-i})$ are stored in look-up table. Input argument in look-up table is just number of iteration i. It can be seen that accuracy is strongly dependent about number of iterations and also, about word length. Since word length will be fixed for whole system, main role in accuracy will have a number of iterations. Number of iterations is limited by clock frequency of device and maximum sampling frequency. In further text there will be discussed accuracy of CORDIC algorithm depending on number of iterations. Sampling frequency of system is 20 MHz, and it is relatively high frequency.

Other solution to calculate arctangent is to use just look-up table. For certain input complex numbers (i.e. its real and imaginary parts) one output is produced. Look-up table is actually a memory which is addressed by ratio of real and imaginary part of complex number and their sign. In memory are stored results for each input combination. Accuracy of this way of calculating arctangent function depends on memory size, i.e. of number of stored results.

If we chose to calculate arctangent function by CODRIC algorithm, it will occupy less resources, but it will take more clock cycles to calculate result. Look-up table calculates result in just one clock cycle, but it will occupy a lot of memory size. Since sampling clock frequency of 802.11a system is large, look-up table is naturally better choice because of speed of calculation, but CODRIC algorithm is necessary because knowing number of iterations facilitates compensation of frequency offset.

After calculating arctangent function, there must be calculated remaining frequency offset, if it exists. By calculating arctangent function, we are only able to estimate frequency offsets in range of -π to π. If frequency offset is bigger, than more operations must be executed. First, received short preamble must be multiplied by original short preamble as $r_2(n) = r_1(n) * p(n)$, where $p(n)$ is original preamble and $r_1(n)$ is received preamble. Both preambles contains a complex numbers. As mentioned before, to multiply two complex numbers three additions and four multiplications are needed. Considering that preamble is 160 samples long, it will take too much time to calculate all that. Other option is to perform 160 multiplications of complex numbers simultaneously.

q , integer frequency offset is defined by maximum value of which is defined as maximum index of $I(q)$ calculated as

$$I(q) = |\sum_{n=0}^{N-1} r_2(n) e^{-j*4\pi qn/N}|^2$$

As it can be seen from above formula, there are also a lot of multiplications needed to calculate additional frequency offset. It is possible to realize only if clock frequency is really high. All these multiplications and additions must be executed within duration of one sampling period.

5.3. SDF and Precedence graph of algorithm

Since it would be too complex to show graph of whole system on one picture, there will be more graphs, each for certain part of system. Also, every of those parts can be considered independently. Signal data flow graph of correlator is shown at figure 5.1.

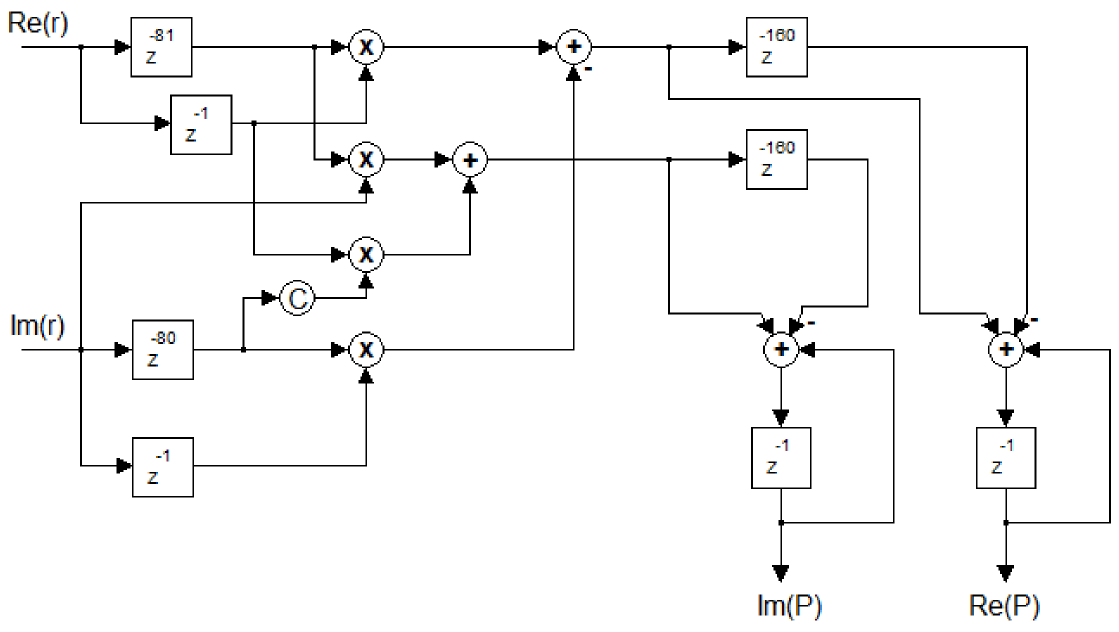

Figure 5.1. – SDF graph of correlator

On the inputs of correlator there are coming samples, on one input real part and on the other imaginary part. As it is mentioned in previous section, samples needed to calculate correlation must be stored, somewhere (80 of them). It is obtained by inserting delay elements into SDF graph which can store 80 samples. Because there must be performed conjugation of complex number, node “C” changes a sign to a delayed imaginary part of a complex number. That is why there are required additional registers because calculating value with different sign consumes some clock time. As it is shown on graph, after certain multiplications, real parts are summed separately, as well as imaginary parts. It is mentioned before that, in order to reduce complexity (less multiplications and additions) are stored in delay elements. It is necessary because last product must be subtracted of total sum of all product and new product must be added to it in order to obtain new autocorrelation function. Result is stored in the buffer. Output of the buffer is connected to input of in order to calculate new autocorrelation value and it forms a loop. Output of correlator is connected to peak detector which determines start of the frame.

Precedence graph of correlator is shown on figure 5.2.

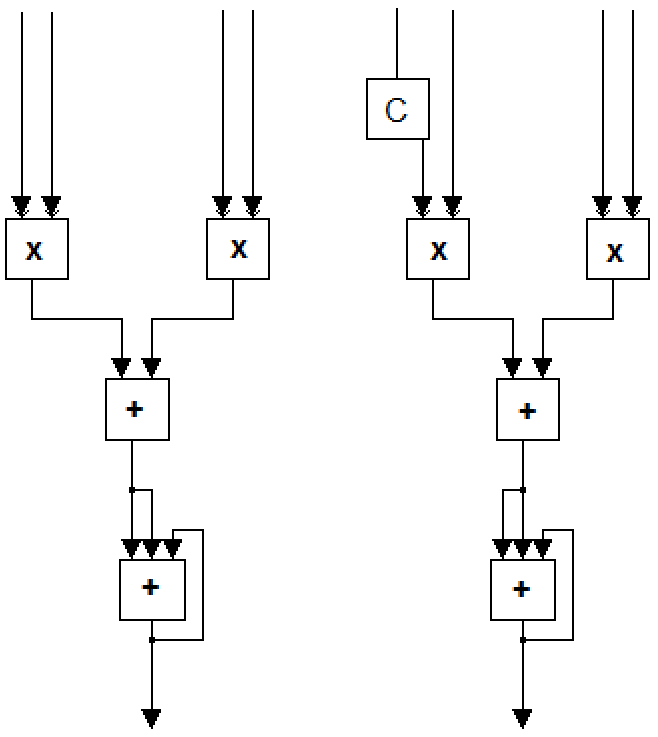

Figure 5.2. – Precedence graph of correlator

Precedence graphs are derived from signal data flow graphs. Aim of precedence graph is to provide better illustration of computational complexity of an algorithm. Precedence graph includes inherent parallelism and computational paths. Since delay elements are not important to show computational complexity of algorithm, they are omitted from the precedence graph.

On graph it can be seen that four multiplications can be performed simultaneously and changing a sign of imaginary part before multiplications. It means also that there are four multipliers necessary to execute all operations at same time. After multiplications, two additions are executed in same time. Next two additions are executed after those two additions because they need results from those operations. Also, it is good to notice that there are two identical parts of the precedence graph. One is used to calculate real part, and other imaginary part of correlation.

There is also necessary to calculate energy of received signal in order to calculate normalized correlation. Within this, it is needed to calculate squared value of received samples. To calculate squared value of complex number it is only necessary to multiply number by itself.

Principle is similar as in calculating autocorrelation. According to formula written above, between two samples there should be executed many additions and multiplications (159 complex multiplications), but there will be used same principle as in correlator. Results will be stored and new sample will be added to result, as well as old will be subtracted from result. Structure of this part is similar to correlator, except there are less delay elements, and there is no need to calculate conjugate complex value, i.e. to change sign of imaginary part of complex number. Data flow graph is shown on figure 5.3.

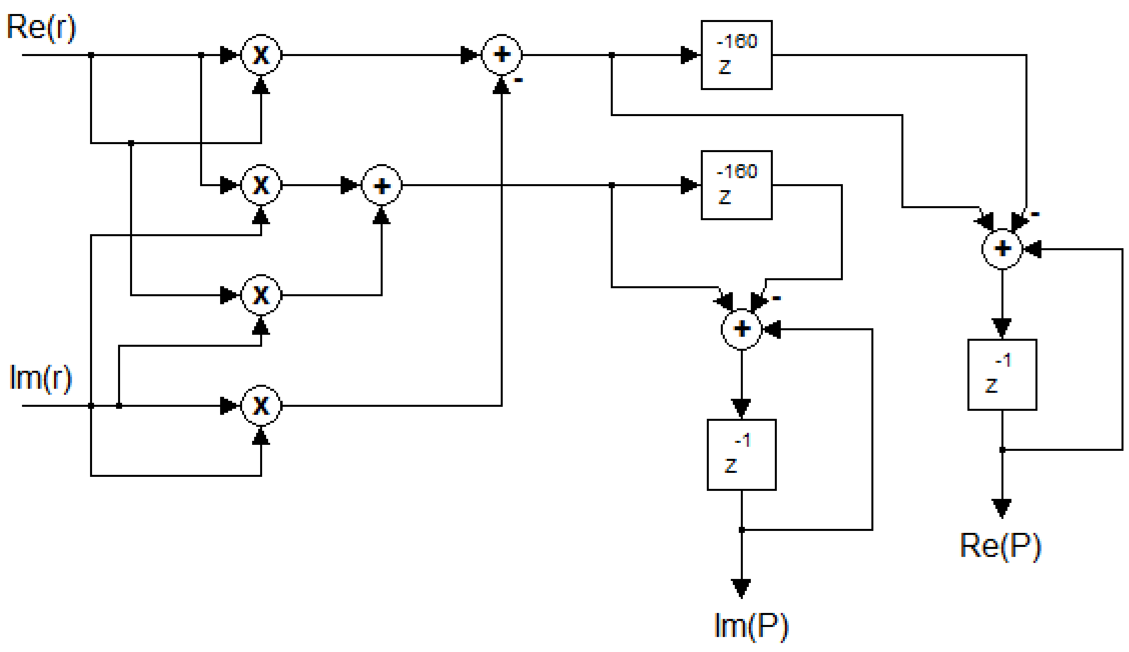

Figure 5.3. – Part for calculating energy of signal

As correlator, this part as a result also produces real and imaginary part of result. In order to reduce complexity of energy calculation, there are two registers which can store 160 samples of input signal. Last one is subtracted from result stored in registers located after adders and first one is added to result and stored in register which works as First In First Out memory (FIFO). One adder is actually subtractor. It actually executes addition except it converts sign of one input parameter, as in correlator as well. Precedence graph of this part is shown on figure 5.4.

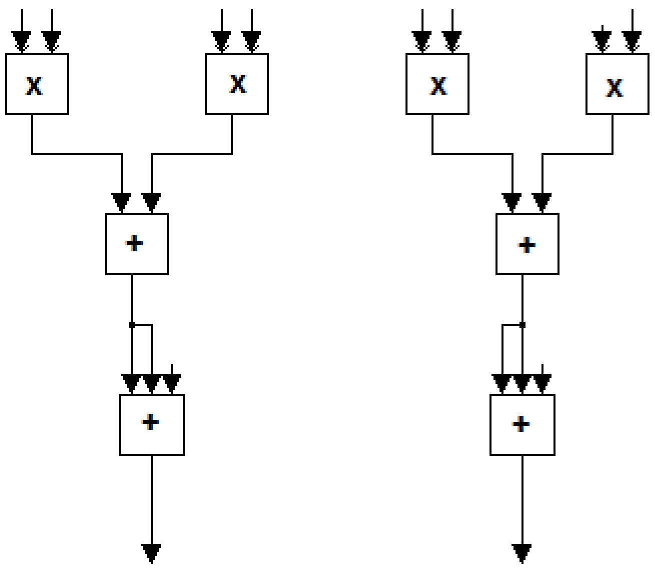

Figure 5.4. – Precedence graph of energy calculator

Four multiplications are executed at same clock cycle. In next clock cycle there are calculated two additions of results, and afterwards, there are in next two cycles calculated subtraction of oldest sample from already stored result and after that new result is added to result stored in accumulator. Adders which are located in part with two clock cycles, are actually two adders each. In first clock cycle one calculates subtraction, and one calculates addition. Possible sharing of units will be discussed in further text.

After correlation and energy of received signal are calculated, next step is to calculate their quotient. Since correlation result and energy result are complex numbers, in order to get real numbers it is necessary to calculate modules of them. It is obtained by squaring real and imaginary part of each complex number and then adding them to each other. In order to achieve this, two multipliers and one adder is needed for each complex number. By each multiplier is calculated squared value of real and imaginary part of complex number and adder to calculate a sum. There cannot be used simplified solution without squaring real and imaginary part since each of them can be negative. Such squared values are brought to input of divider. Divider will be here represented as a block and its structure will not be considered. Digital division is based on Newton-Raphson method. Output of divider is connected to the peak detector which is described in further text. Also, it will be assumed that division by divider consumes three clock cycles. Data flow graph of this part is shown on figure 5.5.

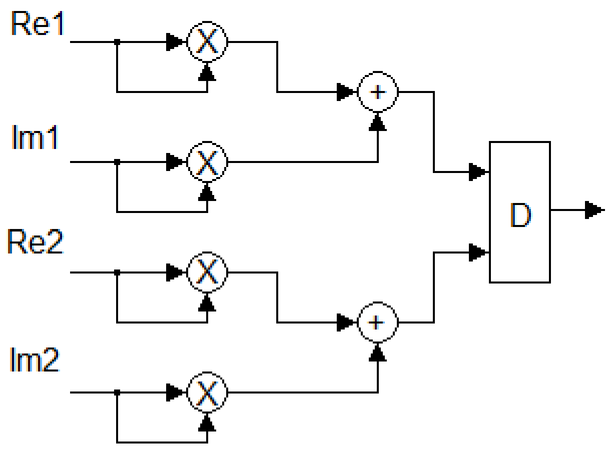

Figure 5.5. – Data flow graph of divider and belonging devices

Precedence graph is shown on figure 5.6. It is assumed that divider consumes three clock cycles to execute division.

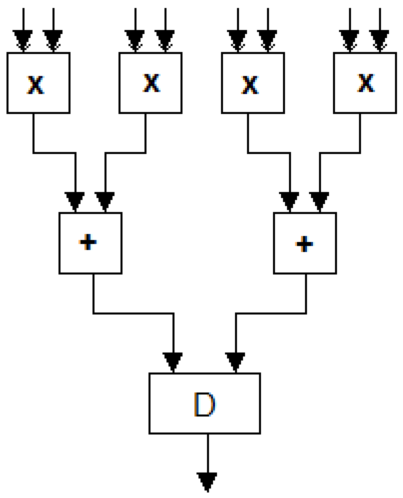

Figure 5.5. – Precedence graph of divider and belonging devices

Next part to be discussed is peak detector. It on its inputs normalized correlation result is coming. Since correlation results which are coming on detector's inputs can be negative, first step is to calculate absolute value of results. To perform this, sign of result must be checked. If the most significant bit is equal to 1, it means that result is negative and must be calculated absolute value of it. It can be obtained by executing inversion operation on a result, and after that adding '1' to result obtained by previous operation. There is needed one device to perform inversion of the bits operation and one for addition. Also, it is required before that to check if it is necessary to make this operation. If using concurrent statements in VHDL language, it is possible to execute all these operations in just one clock cycle.

Calculating of absolute value of complex number is done as

$$|\mathrm{r}| = \sqrt{\mathrm{Re}(\mathrm{r})^2 + \mathrm{Im}(\mathrm{r})^2}$$

Since it would be too complex to implement calculation of squared root of number, it can be easily approximated with

$$approx = |\mathrm{Re}(\mathrm{r})| + |\mathrm{Im}(\mathrm{r})|$$

After calculation of modulo operation, it is necessary to sum real and imaginary part to check correlation result and compare it with threshold. Since there is not root squared operation performed, threshold will not be 0.5 any more. It will be squared root value of 0.5.

Finally, third step is to determine start of the frame. If result of correlation is bigger than threshold, than it will be compared to result which is already stored in the buffer. To do this, there must be used comparators. Comparator compares stored result and new one, and if new one is bigger, it will be stored into buffer. Otherwise, old one will stay in the buffer.

If there is already stored one result of correlation, and if the most recent signal is not bigger than threshold, that means that starting sample corresponds to index of sample stored in the buffer.

Signal data flow graph of peak detector is shown on figure 5.3.

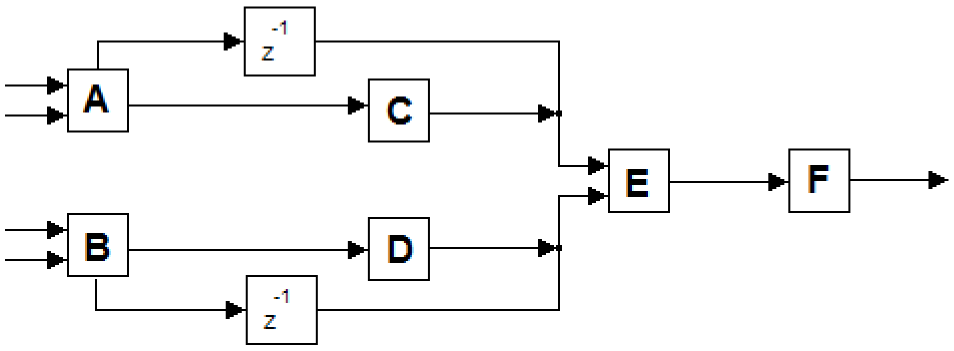

Figure 5.3. – SDF of peak detector

Nodes A and B represents a check units for sign of input values, results of correlations. One node is doing this for real parts, and other one for imaginary parts of result of correlation. Depending on value of most significant bit, there will be or executed calculation of absolute value, or if number is positive, result will be sent to buffer and it will stay there for one sampling period. Buffer is necessary because in the case when one result is negative, and one positive, it will take one more sampling period do calculate absolute value of result with negative sign. During that time, positive result is stored in the buffer. Operations executed in these nodes are conditional.

Nodes C and D calculates absolute value of negative result. To calculate absolute value of the negative number represented in two's complement, it is necessary to invert all bits of input signal and add 1 to it. If inversion is implemented by concurrent statements in VHDL language, both operations (addition and inversion) can be implemented in one clock cycle.

Node E executes addition of real and imaginary part of result and compares result with the threshold. That means one more comparator is needed which compares result of operation with threshold. If result is bigger than threshold, certain signal will be generated on the output.

Finally, node F has a task to determine index of starting sample. If input signal is bigger than threshold, than it must be compared to already stored result in the buffer. If new result is bigger than already stored result, it will be replaced by new one. Otherwise, old one will still be stored in the buffer. If there is already stored result with its index, and new sample is smaller than threshold, start sample is determined.

Precedence graph of peak detector is shown on figure 5.4.

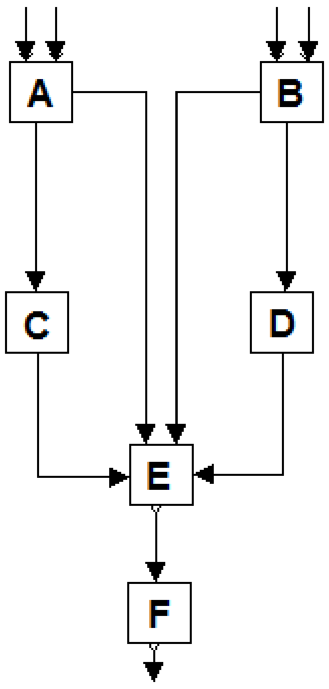

Figure 5.4. – Precedence graph of peak detector

Simultaneously there can be performed two same operations. Checking are real and imaginary parts positive or negative can be executed in same time. Also, in same time can be executed calculations of absolute value if input real and imaginary parts of result of correlations are not positive. After that, addition of real and imaginary parts is executed. It can receive a data directly from input if result of correlation is positive, and from node which calculates absolute value of input data. Selection of source can be executed by multiplexer which is controlled by finite state machine which will be discussed later.

In order to calculate frequency offset, arctangent function must be executed. CORDIC algorithm is already described above, but because its high computational complexity, and also consuming a lot of resources, arctangent function will be calculated using look-up table. Before calculating it, one division must be executed. Argument of arctangent function is ratio of imaginary and real part of complex number, according to

$$\emptyset = \arctan\left(\frac{Im(r)}{Re(r)}\right)$$

Calculation of ratio is not only thing which is necessary to determine frequency offset. Since it is known that arctangent function will only produce results between $-\pi/2, \pi/2$ and results must include range between $-\pi, \pi$, it will be obtained by checking sign of real and imaginary part. If both are positive, result is in range $0\pi/2$, if real is positive, and imaginary negative, result is in range $\pi/2, \pi$, if both are negative then result is in range $\pi, 3\pi/2$, and finally if real is negative and imaginary positive, result is in range $-\pi/2, 0$. As it is said before, there will be used Look-up table to calculate value of arctangent function. Look-up table is actually memory, which is addressed by certain input arguments, and depending on it produces some output. It is obvious that memory must be addressed by three signals. One of them is quotient of imaginary and real part of complex number, and other two are signs of real and imaginary part. All these signals can be merged in one signal and depending on its value, certain output is produced. Memory need to store all values of unit circle, i.e. values in range $0, 2\pi$. Since frequency offset calculated in this part is used to compensate frequency offset, it must be multiplied by received preamble. It is obtained by multiplying signal by $e^{-j2\pi vn/N}$. Complex exponential function is usually calculated as sum of sine and cosine with same argument as complex exponential function. To calculate sine and cosine, dual-port look-up table must be used. How it works in case of sine and cosine, it will be discussed in further text. Also, multiplication of received preamble and exponential function will be considered later, in more details. Procedure is same as multiplication of this result with original preamble. Data flow graph of all these parts is shown on figure 5.5. Multiplier and Look-up table are simplified and more about it will be discussed later.

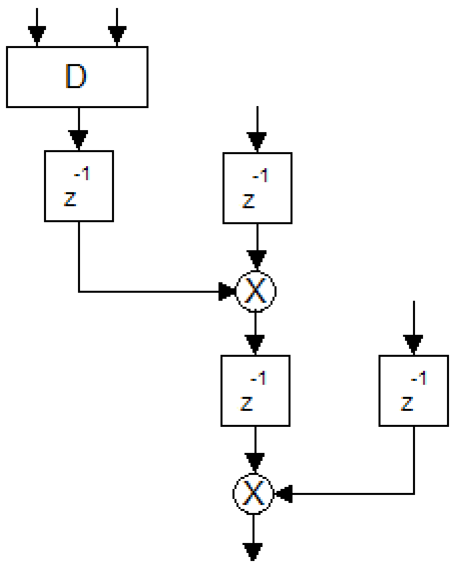

Figure 5.5. – Data flow graph of frequency compensator

Scheme is simplified because multiplier represents complex multiplier which contains four multipliers and two adders. One delay element is actually Look-up table which calculates arctangent function, and other has stored samples of received preamble. Third one contains precalculated values of one part of argument of exponential function. In first multiplier it is multiplied by calculated frequency offset, and in second one result is multiplied by received preamble. Precedence graph is shown on figure 5.6.

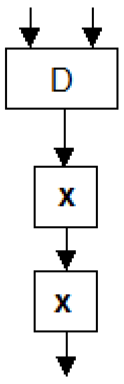

Figure 5.6. – Precedence graph frequency compensator

Possible variants of implementation will be discussed at chapter which deals with scheduling.

After received preamble is compensated by frequency offset, it must be multiplied by original unaffected preamble, as it is mentioned before. Two samples with same indexes must be multiplied and then stored. Simplified precedence graph is shown on figure 5.3.

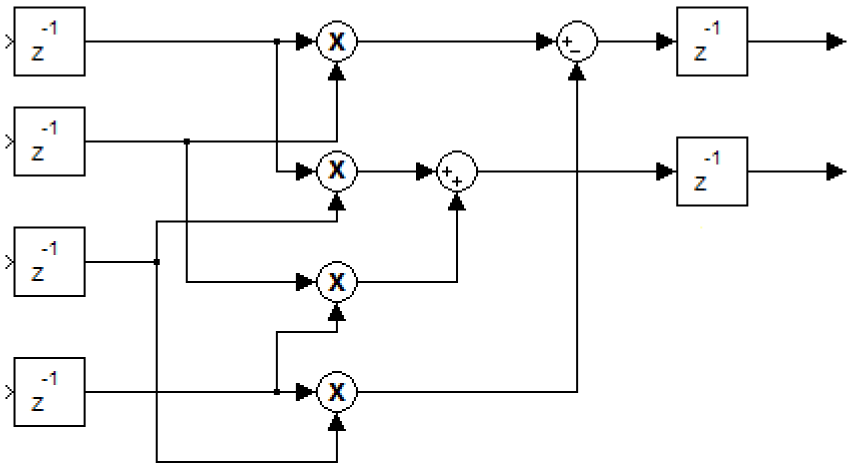

Figure 5.3. – Data flow graph of multiplier of unaffected preamble and received preamble

As in correlator, there are four multiplications and two adders needed per one complex multiplication. Samples of received preamble which need to be multiplied by original unaffected preamble are stored in a buffer which can contain 320 samples, 160 for real, and 160 for imaginary part of complex number. Also, there is one buffer needed to store unaffected preamble which has same capacity as buffer in which is stored unaffected preamble. These memories are represented as delay elements in signal data flow graph. On the output there are also two delay elements. They represent a buffer where results of multiplications are stored. Signal data flow graph shown in figure 5.3. is actually simplified graph because it represents just one of 160 complex multiplications. To represent all, there should be 160 figures as this one.

Precedence graph derived from simplified signal data flow graph is shown in figure 5.4.

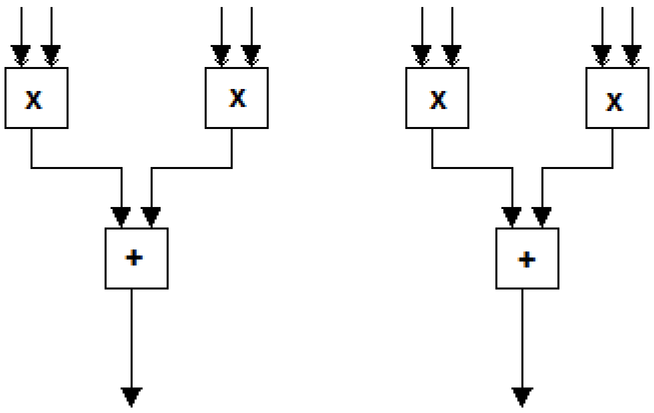

Figure 5.4. – Precedence graph of multiplier of unaffected preamble and received preamble

As in correlator, there are at least four multiplications executed in same time as well as two additions after multiplications are executed.

Part which will be discussed in this part which calculates $I(q)$ according to this equation

$$I(q) = |\sum_{n=0}^{N-1} r_2(n)e^{-j*4\pi qn/N}|^2$$

This part will be considered in two separate parts. One is part which calculates part of the formula related to exponential functions. This includes calculating input argument of exponential function and calculating result of exponential function. Exponential function can be calculated as sum of sine and cosine with same input argument for sine and cosine as for exponential function. Part of equation $e^{-j*4\pi qn/N}$ by sine and cosine is represented as

$$e^{-j*4\pi qn/N} = \cos(4\pi qn/N) - \mathrm{j} * \sin(4\pi qn/N)$$

As it can be seen, cosine part of complex exponential function represents real, and sine part represents imaginary part. It those all multiplications and divisions will be calculated for each argument, it will significantly increase complexity of the system. It is known that 4π, q, n and N are constants. Taking this into account, $4\pi qn/N$ can be calculated before and stored in memory and addressing memory with certain parameters (in this case q and n). It is necessary to mention that q and n are vectors with constant numbers as it can be seen from formula mentioned before in the text. This solution is much better than calculating all time input arguments for sine and cosine especially because there are always same results present. It will consume more memory and storage resources than if it would be calculated constantly, but it will not use any adder or multiplier for calculations which are more complex than storage elements. To get results of sine and cosine there will be used procedure called Direct Digital Synthesis (DDS). It uses Look-up tables which are actually memories. They are dual-port memories, which contains data which represent amplitudes of sine and cosine wave. They are generating it on their outputs depending on input which is actually address. So, it can be seen that in this procedure argument of sine and cosine is address and depending on it there will be produced certain output which represent amplitude, i.e. sine and cosine value of certain argument. Accuracy of sine and cosine is determined by amount of data stored in memories. As memory is larger, it is more accurate, as well as data length of argument because it need to be able to address all memory locations. Only one address is needed to get result of sine and cosine and how it is obtained will be discussed in further chapters. Also, it is just required to store first quarter of sine in memory in order to get full period of sine and cosine. It will also be explained in further chapters. Data flow graph of this part is not shown as other parts because it contains only two storage elements. First storage element contains already calculated constants for each q and n. With these values output is controlled and those values are generated by finite state machine what will be discussed in further chapters. Output of this storage element is controlling output of Look-up tables which are producing sine and cosine signals.

This part multiplies result of multiplications of unaffected preamble and received preamble with exponential function as it is shown in formula above. Calculation of argument of exponential function is discussed in previous paragraph. After multiplication of each element with certain result of exponential function, they must be summed and squared. This process must be repeated for each q where $q = \{-\frac{N}{4}, \dots, -1, 0, 1, \dots, \frac{N}{4}\}$. It means that this process must be repeated 81 times. It will consume a lot of clock cycles obviously, and how much, it will be discussed in next chapter. After all operations are executed, there must be found biggest result, i.e. index q for which result is biggest. Signal data flow graph is shown in figure 5.5. It is simplified because each multiplier in this case represents one complex multiplication. It has a same structure as devices shown in figures 5.1 and 5.3.

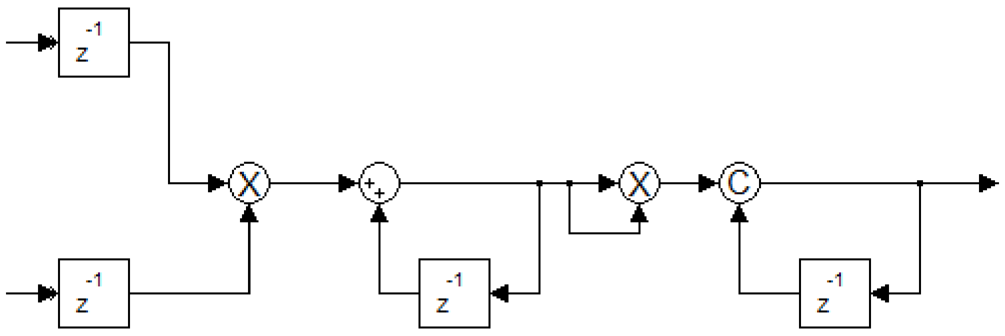

Figure 5.5. – Data flow graph

Results are stored in buffers (real and complex part) and they are multiplied for certain q. Then they are summed and it is iterative process so there is one iteration loop. It can be solved on different way, but it will be discussed in scheduling chapter. Next step is to calculate squared value. Multiplier's inputs are connected on same line and result of this operation is brought to comparator. It is compared here to old results and the biggest one (and its index) is stored and later used for compensation.

Precedence graph is shown in figure 5.6.

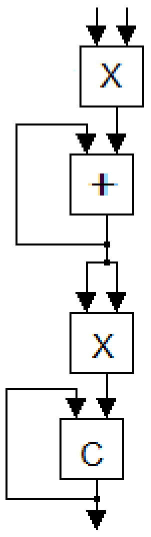

Figure 5.6. – precedence graph

From precedence graph are ommited delay and storage elements.

5.4. Scheduling, allocation and assignment

Scheduling is process of scheduling all operations in the cycles. It means that to each operation must be assigned a certain amount of time to execute. Depending on operation type, it can consume one or more clock cycles to perform the task. Allocation is process which defines type and number of hardware resources. This includes various functional units (adders, multipliers, multiplexors), storage elements such as buffers and ROM memories, and buses (connectivity elements). Scheduling and allocation can be made by a sort of transformation of precedence graph as shown in figure 5.5.

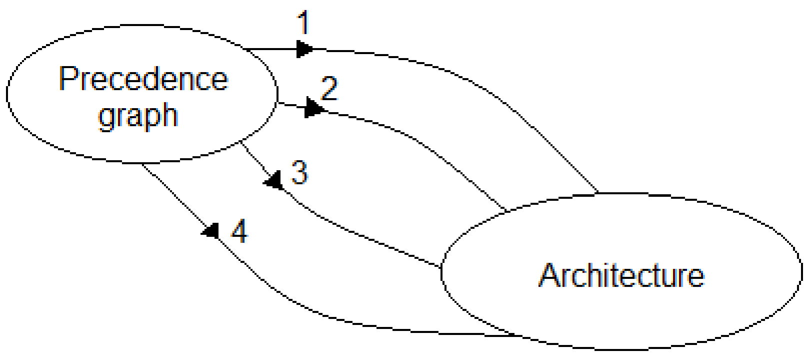

Figure 5.5. – Transformation of precedence graph to architecture

Numbers on the lines are:

1. Scheduling
2. Allocation
3. Assignment
4. Control design

In this chapter, scheduling and allocation are going to be discussed. There are two ways to do this and it depends on constraints. If there is a limited amount of resources available, than first to be done is allocation and after that scheduling. Aim is to compose such architecture that as less as possible time is needed to execute all required operations. Second case is when system is time constrained. In this case, scheduling is first what must be done, and next step is than allocation. Aim is to use as less as possible resources to create device which is available to execute all required operations in given time.

In this project, system is time constrained. It is because sampling frequency of system is already given by system specification standard.

Scheduling of correlator is shown on figure 5.6.

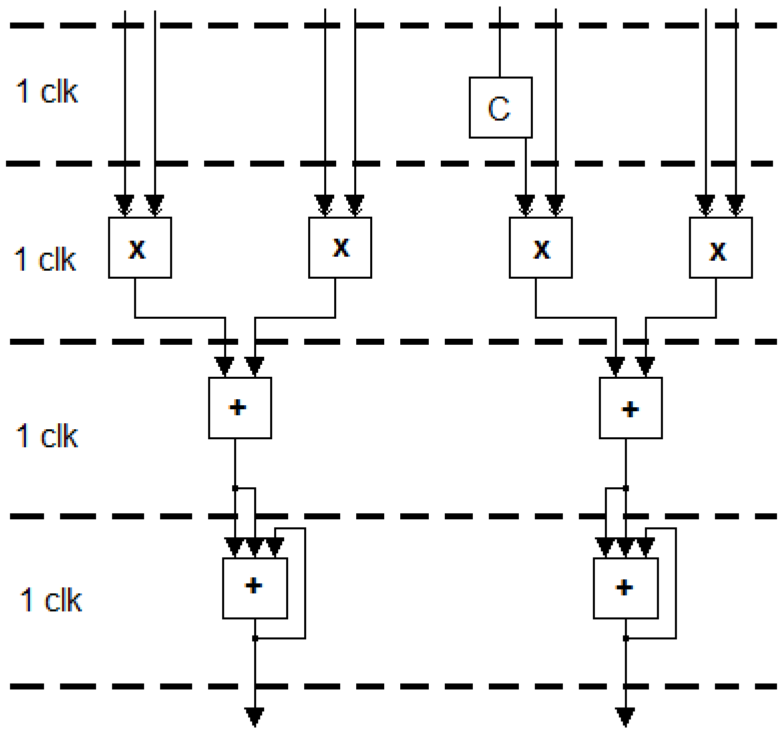

Figure 5.6. – Scheduling of correlator

As it is mentioned in previous chapter four multiplications are executed simultaneously, and two additions. To execute each of these operations, one clock cycle is needed. Before multiplications, there must be changed a sign of imaginary part and it consumes one clock cycle. So, in four clock cycles everything can be executed.

Scheduling of energy calculator is shown on figure 5.7. It is actually very similar to scheduling of correlator, except it does not contain part to change sign of one imaginary part which consumes one clock cycle.

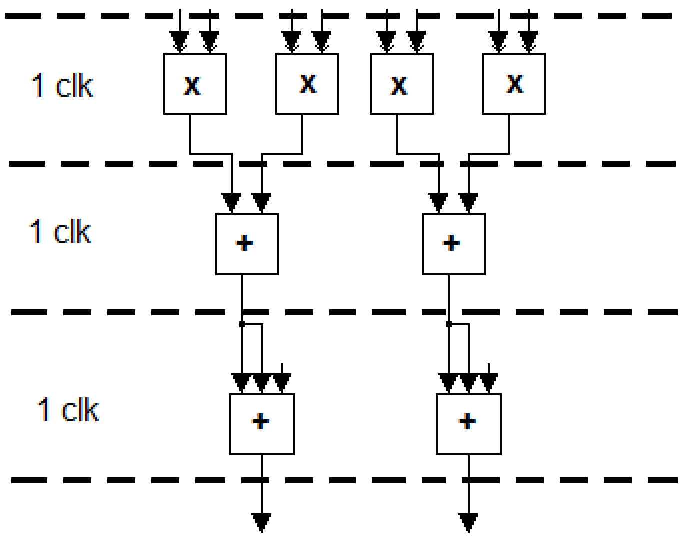

Figure 5.7. – Scheduling of energy calculator

According to figure 5.6. there are four multipliers and four adders needed to realize correlator. Four multipliers are required because four multiplications must be executed in same time. Also, four adders are also required to execute two additions in each clock cycle. Sharing of resources, in this case adders would be possible, but there is necessary to introduce one more register which will store result of previous addition. Beside this, as can be seen on figure 5.1., there are some storage elements needed. Two buffers which has capacity of 80 samples, whose purpose is to store 80 samples to calculate correlation function, are needed. Another two buffers with 160 samples capacity in order to store 160 result of additions. Their presence significantly reduces computational complexity of correlation, as it is explained in chapter 5.1. Two registers need to be able to store just one sample of real and one of imaginary result of correlation. Finally, all multipliers and adders has to have one register on their output to store the result of additions and multiplications. Total number of functional and storage elements for correlator is in table 5.2.

Type	**Amount**
Multiplier	4
Adder	4
Register	10
Buffers	4

Table 5.2. – Number of functional units and storage elements in correlator

As in previous chapter is mentioned, after energy calculator and correlator, next device in datapath is divider and belonging devices which calculated squared vaules of results of those two parts. Scheduling is shown on figure 5.8.

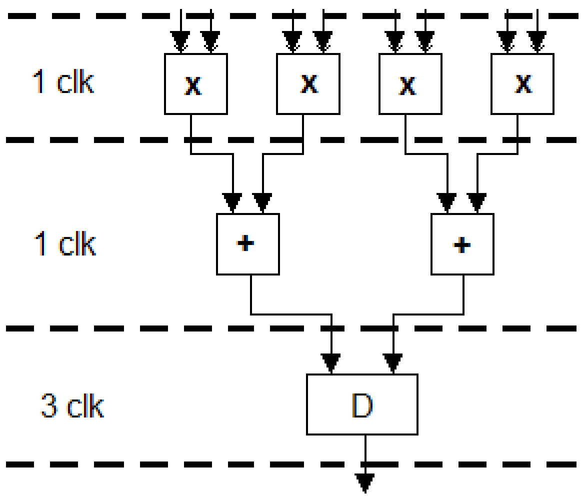

Figure 5.8. – Scheduling of divider and belonging parts

Devices used in this part are listed in table 5.3.

Type	Amount
Multiplier	4
Adder	2
Register	7
Divider	1

Table 5.3. – Number of functional units and storage elements in divider

Scheduling of peak detector is shown on figure 5.7. Nodes A and B are comparators. They are checking is the sign bit ‘0’ or ‘1’, and depending on this result, data will be sent to the buffer or to the nodes C and D. It consumes one clock cycle. Nodes C and D are calculating absolute value of correlation result. These nodes are containing inverter and adder. To invert signal it takes only as much as it is delay time of inverter. In total, one cycle is needed to execute this operation. Node E is adder and it takes one clock cycle to execute operation, and also it compares result of operation with threshold (also as in previous case, it takes as much as delay time of element is). Node F is finally checking is result of correlation bigger than already stored result. It takes one clock cycle. Depending on result of comparison, data will be written in buffer, as its index. Otherwise, old data will remain at the buffer.

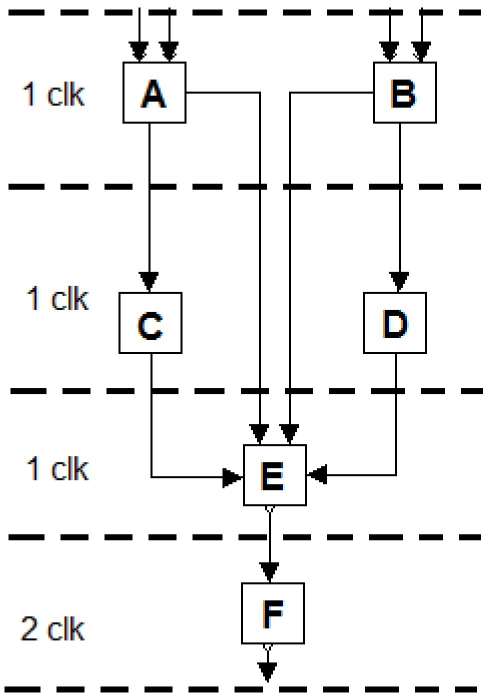

Figure 5.7. – Scheduling of peak detector

Referring to a scheduling of each device, there must be determined which devices will be used to each part. Also, there will be considered possible sharing of resources as part of assignment process.

Peak detector must contain two comparators to check sign bit of results of correlation. Depending on result, data will be send to buffer or to node C or D which calculates absolute value It will be controlled by finite state machine, which is discussed in next chapter. Also, to store data temporally, one register must be added to comparator. One clock cycle is required to execute this. To calculate absolute value, two adders are needed. On their inputs there must be a device which makes inversion of each bit. To execute this operation, one clock cycle is required. As in case of previous adders, register must be on its output to store result. One more adder (and register) is required to execute addition of real and imaginary part. On the output of adder there will be a connected a comparator which generates ‘0’ or ‘1’ depending on result of addition. If it is bigger of threshold, it will generate ‘1’, otherwise, it will generate ‘0’. These signals are actually a control signals for comparator which determines starting sample which enables or disables device to execute operation. Two clock cycles are required do execute all operations in node F. First one is comparison of new data and already stored data and this operation requires one clock cycle to execute this. Other clock cycle is needed to store new result and new index in the buffer. One buffer is required for comparator, one to store result and one to store index. Functional units and storage elements of peak detector are in table 5.3. There is not possible to share any functional units without adding more registers or buffers. There is also not possible to make any sharing of units between peak detector and correlator without more registers. In each clock cycle every functional unit executes operation in correlator as well as in peak detector.

Type	Amount
Adder	3
Register	9

Table 5.3. – Functional units and storage elements in peak detector

After start sample is determined, frequency offset must be calculated. In figure 5.8. scheduling of this part as well as frequency compensator will be shown.

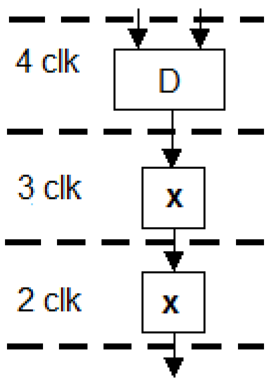

Figure 5.8. – Scheduling of frequency compensator

Since delay elements (memories and registers) must be omitted from precedence graph as well as from scheduling, one clock cycle is added to divider because it consumes three clock cycles to execute division, and one cycle is added because one clock cycle is needed to get arctangent function result depending on result of division and signs of input arguments. As it is said in previous chapter, one complex multiplication must be executed because calculated frequency offset must be multiplied by constant $2\pi n/N$ which is stored in one memory, and second multiplication must be executed to get result of multiplication of received preamble and exponential function which compensates frequency offset. As it will be explained later, exponential function is calculated as sum of sine and cosine. Here it will be assumed that all 160 results are generated in same time, i.e. it consumes just one clock cycle to calculate sine and cosine depending on its result. Also, one clock cycle is added to first multiplier because it is needed to calculate sine and cosine by look-up tables which are modeled as delay elements and they are omitted from precedence graph and scheduling.

Scheduling of multiplier of unaffected preamble and received preamble will be considered in more ways. All 160 complex multiplications can be executed in same time. This will require 640 multipliers and 320 adders, but it will consume only two clock cycles to execute 160 complex multiplications. One complex multiplication can be realized by less than four multipliers and two adders, but it will require additional registers to store intermediate results. Saving on resources is not obtained, but time of execution has been prolonged. Scheduling of this part using 320 adders and 640 multipliers is shown on figure 5.8.

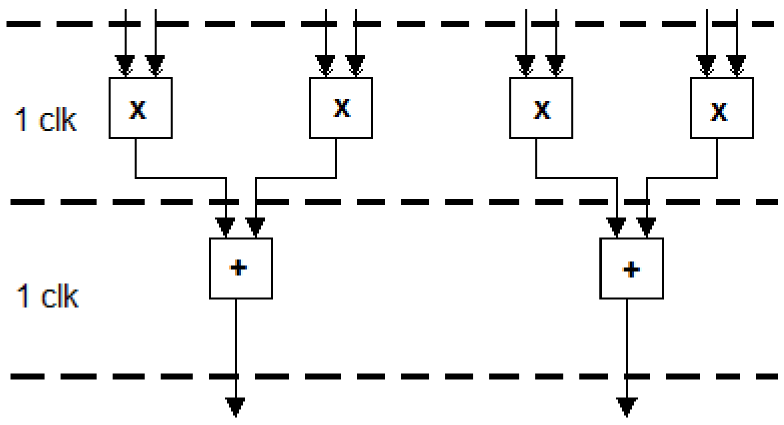

Figure 5.8. – Scheduling with 640 multipliers and 320 adders

In contrast, if there will be used only four multipliers and two adders to execute 160 complex multiplications. It will consume 320 clock cycles. Scheduling of this case is shown in figure 5.9.

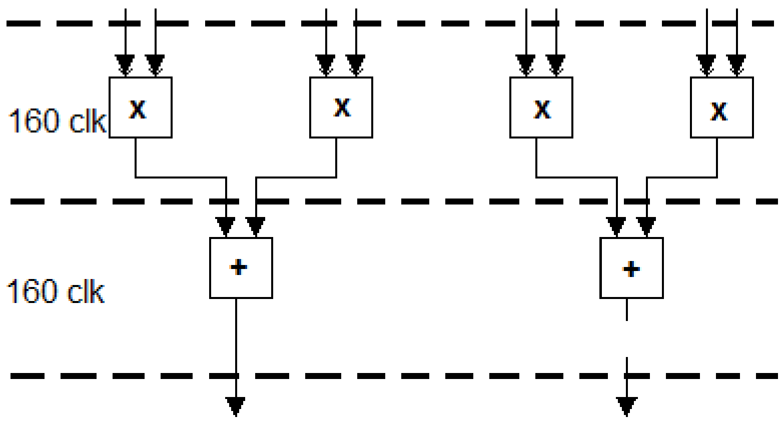

Figure 5.9. - Scheduling with 4 multipliers and 2 adders

Compromise between computational time and used resources is shown on time-resources diagram on figure 5.10. Number of resources includes number of adders and number of multipliers.

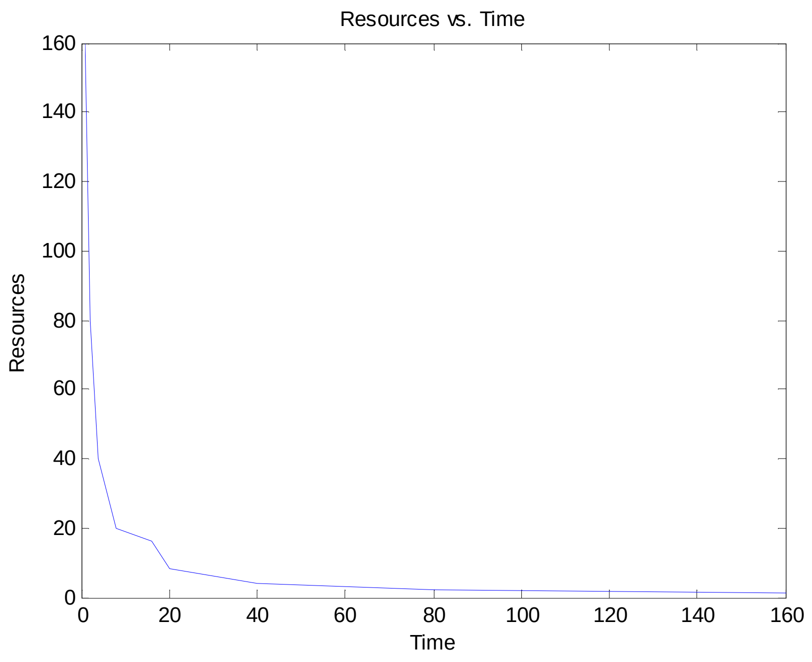

Figure 5.10. Execution time depending on adders and multipliers

As it is said before, results of these multiplications are used for another multiplication. Number of resources for these multiplications will be chosen depending on time needed to calculate results of other multiplication which are multiplied by these results. Analysis of time needed to calculate those multiplications is analyzed in further text.

Next part to be considered here is a part which multiplies result of multiplications of two preambles with exponential function whose arguments are calculated in device described above. There are several possible solutions how to schedule this part. Also, it will be closely related to part which calculates multiplication of unaffected preamble and received preamble. One solution is fully parallel and it is shown on figure 5.11.

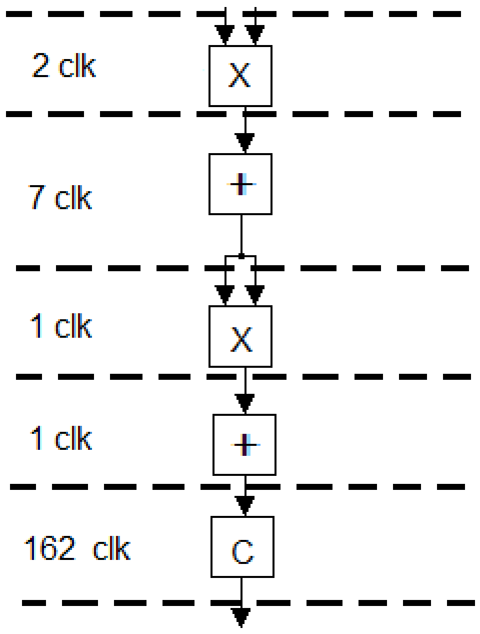

Figure 5.11. – Scheduling of fully parallel solution

There are two clock cycles needed to execute complex multiplication. It will use 160 complex multipliers. Next step is to calculate sum of results of multiplications. By using fully parallel solution, there are seven clock cycles required to calculate sum of all results of multiplications for each q. If all complex sums for each q are executed in same time and fully parallel, that will require 308 adders for each q. In total, it is 24948 adders. Next step is to calculate squared value of each sum. There is one value for real and one for imaginary part of complex number and each of them must be squared. It is obtained using two multipliers per each q. Since there are 81 results for each q in fully parallel solution there must be used 162 multipliers. Afterwards, it is necessary to sum real and imaginary squared part of complex number. It uses just one adder and consumes one clock cycle. Final step is to compare all 81 result and chose for which q result is biggest. One clock cycle is used for computing, and one is used for storing, so in total there are 162 clock cycles needed to determine biggest value.

Other solution is fully serial and is considered in this paragraph. It will use one complex multiplier (i.e. four multipliers and two adders) to calculate product of unaffected (original) preamble and received preamble. Each complex multiplication consumes 2 clock cycles. There are two clock cycles needed to get a first result of multiplications, so there is in total required 161 clock cycle to get results of all multiplications since while calculating sum of first two samples to get result, there are simultaneously calculated results of multiplication of second two samples, etc. Next step is to calculate sum of multiplications. If we consider this solution as fully serial, it means only one adder and one register will be used to calculate sum for each q. If one register is used it means that 160 clock cycles must be used to calculate sum for each q. In total, since there are 81 values for each q in this fully serial implementation it means that it will consume 12960 clock cycles to calculate sum of all q. It is also necessary to determine q for which sum is biggest. One cycle is needed for comparison and one is needed for storing value, as in fully parallel implementation. Scheduling of fully serial solution is shown on figure 5.12.

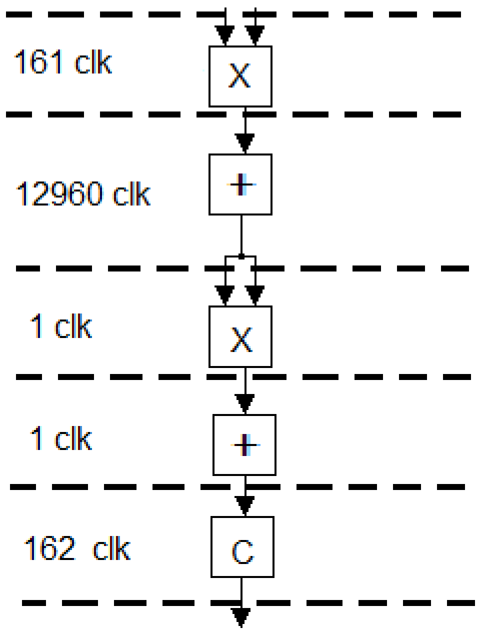

Figure 5.11. – Scheduling of fully serial solution

There can be also considered third solution which can be called mixed because it contains some parts which are implemented as fully serial, and some which are implemented as fully parallel. Since last three parts are same in fully serial and fully parallel solution, there are just two combinations for mixed solution. One is to use part for multiplication from fully parallel solution and additions from fully serial solution. Second one is to use addition part from fully parallel solution and part for multiplications from fully serial solution. Resources vs. time diagram is shown on figure 5.12. It includes all cases, fully parallel, fully serial and two solutions for mixed solution.

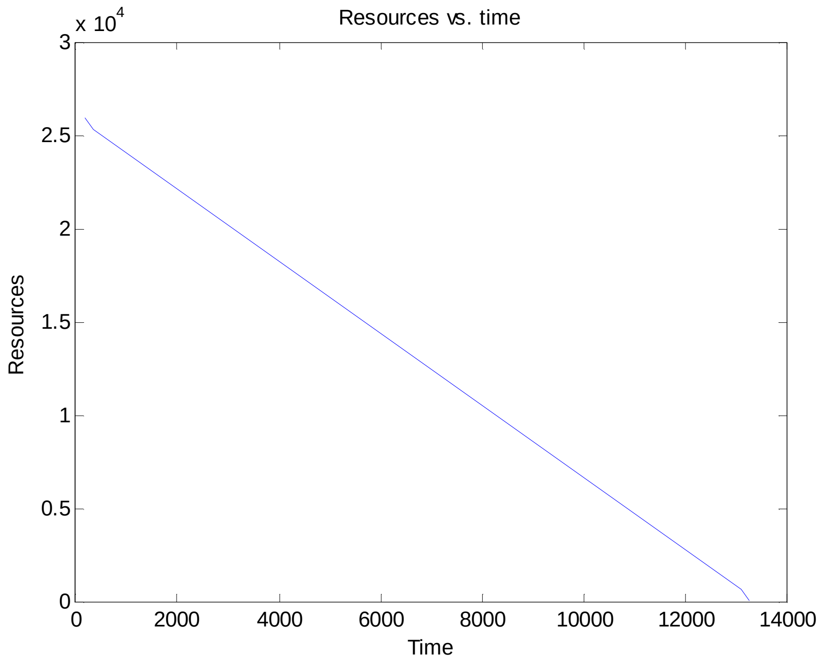

Figure 5.12. – Time vs. resources diagram of all solutions

In total, for fully parallel solution there are needed 25269 adders, 642 multipliers, one comparator, one register to store temporal result, one to store index q, two registers to store multiplication results of original and received preamble and multiplication of other parameters, mentioned when describing previous device. In fully serial solution, everything is same as in fully parallel solution except number of adders and number of multipliers. In this solution there are used six multipliers and five adders. Other two mixed solutions also have all same as in previous cases except number of multipliers and adders. One solution has 25269 adders and six multipliers, and other one has 642 multipliers and five adders.

5.5. Control design

To describe function of device, there have been used signal data flow graph and precedence graph. After scheduling, there are known all devices which must be used to realize this device, and connections between them as well. However, it is not enough for device to perform its function. All these functional units, storage elements and buffers and their work must be controlled. This is obtained using finite state machine (FSM). Finite state machine consists of a set of a states, a set of transitions from one state to another, and set of outputs. There are two main types of finite state machines: Mealy and Moore. Mealy machine produces certain output depending on current state of machine and input value. In contrast, Moore machine output depend just on current state. Mathematically, it can be represented as

$$\langle S, I, O, n: SxI \rightarrow S, o: SxI \rightarrow O \rangle$$

S is a set of the states, O is a set of the outputs, and I is set of inputs. n represents next state logic (calculates next state depending on current state and input), and o is output logic which calculates output depending on current state and input. This is in the case of Mealy machine. In case of Moore machine, it can be represented as

$$\langle S, I, O, n: SxI \rightarrow S, o: S \rightarrow O \rangle$$

because output depends just on current state, not on input also. General scheme of finite state machine is shown on figure 5.8.

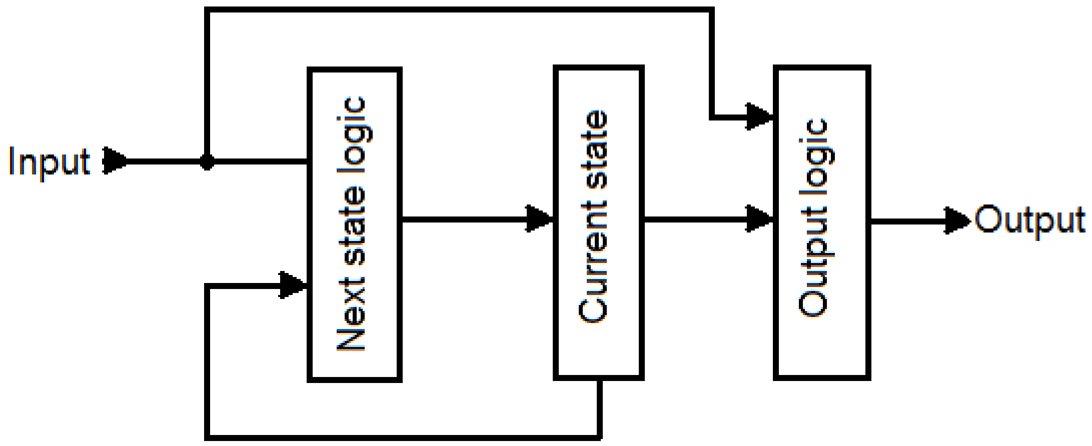

Figure 5.8. – Finite state machine

In this figure is shown Mealy finite state machine. If it would represent Moore state machine, than input should not be connected to output.

Device in this project actually consists of two independent parts, since certain amount of time is just one active, and after that certain amount of time other part is active. So each part and its state machines will be considered independently. Only connection will be activation of second part by first part.

There will be one state machine for correlator and peak detector, as well as for energy calculator since it performs similar operations in same time as correlator. Those three subparts makes first independent part. Finite state machine of this part need to have three states. First state, which will be denoted as s_1 is when device is turned on, i.e. it starts to work and in this state machine stays only once. While there are not 160 samples processed, it will be assumed that initial 160 samples exist and they are equal to zero. Finite state machine will change its state when 160 samples are processed by correlator. Obviously, there is one counter needed. This state is denoted as s_2. That is because peak detector does not need to perform any operation if there are not enough samples processed. In this state finite state machine will activate peak detector. It will stay in this state as long as peak detector detect start of the frame. After that, correlator and peak detector do not need to perform any function and becomes idle (this state will be denoted as state s_3). All this can be described by using ASM chart (Algorithmic State Machine chart). It can be considered as one way to design finite state machine, and also it can be considered as less formal version of finite state machine. Unlike finite state machine, algorithmic state machine includes also variables used in datapath.

It is mentioned that correlator and energy calculator are executing their functions in same time, and they and their variables can be analyzed in same time. Possible units, register and bus sharing will be considered later. State machine which controls correlator and energy calculator has three states. Last state is when those devices are idle. Variables in datapath are same during first two states. Changing of state actually does not have any influence on correlator and energy detector because it only activates peak detector after 160 samples are processed. Only difference will be that there will no more be initial samples, which has values equal to zero. In correlator and energy calculator samples are all the time coming in, being processed, and going out, just several of them will be taken to analyze correlator and energy calculator. According to precedence graph and data flow graph of correlator, there are four input variables. One variable, which is not on graphs, is control variable and its purpose is to change state. It is actually output of one counter which is counting how many samples are processed. Input variables in correlator can be denoted a1, a2, b1, and b2. a1 can be considered as real part of first complex numer, a2 as imaginary part of first complex number, and b1 and b2 as real and imaginary part of second complex number. Imaginary part of one complex number must have sign changed. Meanwhile, other variables are stored in registers for one clock cycle since one clock cycle is needed to execute this operation. This variable can be denoted as b2'. Outputs of the multipliers are calculated in next clock cycle and they can be denoted as m1, m2, m3, and m4. Outputs of adders which has inputs outputs of multiplier can be denoted as d1 and d2. There are two more adders in this device. Inputs in first ones are variables d1 and d2, and some old results which are calculated 160 clock cycles before, and they can be denoted as d3 and d4. There is actually subtraction performed, as it is described in previous chapter. Results of subtraction can be denoted as e1 and e2. Then it is added to result already stored in register, as it is described in previous chapter. Those variables can be denoted as e3 and e4 and their result as f1 and f2. Life time analysis of variables is shown in table 5.4.

	Cl1	Cl2	Cl3	Cl4	Cl5	Cl6
a1, a2, b1, b2	X	X				
b2'		X				
m1, m2, m3, m4			X			
d1, d2, d3, d4				X		
e1, e2, e3, e4					X	
f1, f2						X

Table 5.4. – Life time analysis of correlator variables

In this analysis only one set of variables is used, i.e. only one sample is considered. All these variables are present during first two states of finite state machine. To get a better view, equations will be written in table 5.5.

m1=a1*a2	m2=a1*b1	m3=a1*b2’	m4=b1*b2’
d1=m1+m3	d2=m2-m4	e1=d1-d3	e2=d2-d4
f1=e1+e3	f2=e2+e4		

Table 5.5. – Equations in correlator

Table 5.5. is enclosed here because it can produce a good view in similarities between correlator and energy calculator. Unlike correlator, energy calculator has just two inputs, but it has also four multipliers as correlator. Energy calculator calculates in first step squared value of complex number, so some multipliers has same inputs. Some inputs are also same as inputs in correlator. Let’s take that a1 and a2 are same as in correlator. Equations in energy calculator are shown in table 5.6.

M1=a1*a1	M2=a1*a2	M3=a1*a2	M4=a2*a2
D1=M1+M2	D2=M3+M4	E1=D1-D3	E2=D2-D4
F1=E1+E3	F2=E2+E4		

Table 5.6. – Equations of energy calculator

Similar as in case of correlator, with M1, M2, M3, and M4 are denoted outputs of multiplier. D1 and D2 are outputs of adders, E1 and E2 are next two adders which actually execute subtractions, and finally F1 and F2 are adders that add this value to final result. Life time analysis is shown in table 5.7.

	Cl1	Cl2	Cl3	Cl4	Cl5
a1, a2	X				
M1, M2, M3, M4		X			
D1, D2, D3, D4			X		
E1, E2, E3, E4				X	
F1, F2					X

Table 5.7. – Life time analysis of energy calculator

From first table it can be seen that two multipliers has same input variables, and it means output also. Obviously, one multiplier from energy calculator can be obtained. Only one more bus must be added to output of multiplier to connect it to adders. Also, one multiplier of correlator and one in energy calculator has same inputs (and also output). Since correlator and energy calculator executes their operations in same time, one multiplier from one of this devices can be omitted and instead of it used certain multiplier in other device, i.e. share it. It is just necessary to add one more bus to connect multiplier's output of one device to adder's input of other device. Also, some adders could be shared as well. I.e. when variables D1-D4 are calculated, those adders can be used to calculate variables F1 and F2 in energy calculator. Hurdle can be if immediately after that addition is executed, new samples are coming on adder's inputs because then it cannot be used for other variables. This analysis is not actually real life time analysis since it is made depending on clock cycles, and not depending on states since there are only two active states in this case.

Divider and its belonging parts has also similar structure to correlator and energy detector. It has four multipliers. Both inputs in each multiplier are same since aim is to calculate squared value of it. Those variables can be denoted as r1, i1, r2 and i2. Results of multiplications are inputs of two adders. Outputs can be denoted as a1 and a2. These values are not same values as those in correlator. Those values are inputs in divider. It is possible that divider uses multipliers and adders from correlator or energy calculator only in the case that clock frequency is high enough, i.e. there are no need for other data to be processed by correlator or energy calculator. If certain data is processed in same time by correlator or energy calculator and divider, there is not possible to share functional units (in this case adders and multipliers).

As it is said earlier, state machine which controls this part has three states. They are controlled by two signals. One is output of the counter which generates signal when 160 samples are processed. Second one is output of peak detector. When peak is detected, it transforms finite state machine in third state, in which is actually device idle. State diagram is shown on figure 5.9.

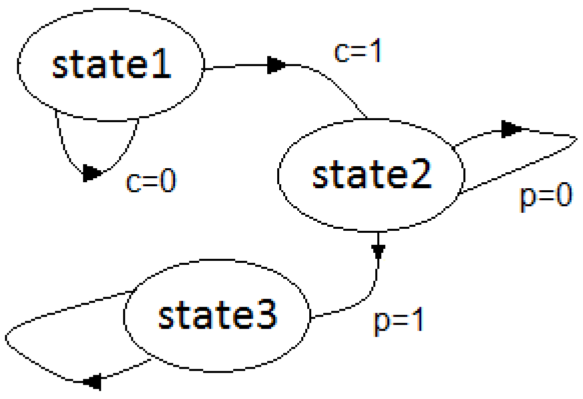

Figure 5.9. – State diagram of first part

Signal which comes out from the counter is denoted as ‘c’, and signal which comes out from peak detector is denoted as ‘p’.

Second part of device deals with frequency offset estimation and it consists of three main parts. First one calculates frequency offset, and compensates affected preamble with it. Second part multiplies result of this part with unaffected (original) preamble. Third part finally determines index q for which value I(q) is biggest. In frequency offset estimator first part is first which executes its function, second is second, and third executes it as last one. First part contains one divider, buffer which stores received preamble, one buffer which has precalculated values for argument for exponential function and look-up tables in order to calculate sine and cosine. First part of receiver has already finished its function so its devices can be used later for some other function. Divider is needed in first part of frequency estimator, and it is not used any more so this functional unit can be shared. It also contains two sets of multipliers. One set multiplies frequency offset by constants stored in memory, and other is set of complex multipliers which multiplies received preamble by exponential function, i.e. by sine and cosine with same arguments. When it is calculated, than second part must be activated. It multiplies earlier calculated result by original preamble. It uses same set of multipliers as part which multiplies received preamble by exponential function. So, another functional unit can be shared. Result can be stored in register which earlier stored received preamble since it is not necessary any more. This result also must be multiplied by exponential function in the third part. Again, there are needed look-up tables to generate sine and cosine which are used instead of exponential function. Same type of complex multiplier is used, and is shared by all three parts of the frequency estimator.

6. Conclusion

This paper deals with acquisition methods of OFDM system. Those methods has a purpose to detect start of the frame or symbol, and also to estimate frequency offset. All these operations are done before FFT (Fast Fourier Transform). Some facts known even before have been proven through some simulations made in MATLAB. It was proven that OFDM signal is very sensitive to sampling frequency offsets between receiver and transmitter, and also to carrier frequency offset introduced in channel or during frequency mismatch of oscillators in receiver and transmitter. Sensitivity is proven also to synchronization, and it was a deal of one part of this paper. There were analyzed methods for timing synchronization and for frequency offset estimation. There were chosen several methods and some of them had both, and timing estimation and frequency offset estimation. Timing synchronization was analyzed separately from frequency offset. Criteria of choosing methods was that they must be applicable to 802.11a standard. Parameters which were analyzed were probability of accurate detection, variance and mean squared error. In frequency offset analysis probability was not considered, but other parameter which was considered was range of estimation. Renn method showed best results according to all parameters, especially in range of estimation.

In architectural design attention was paid to using as less as possible resources but that all timing constraints are satisfied. Significant reduction of complexity in the correlator was obtained with one accumulator which stores old results, and just with two adders (one addition and one subtraction) calculates new result of correlation instead of calculating multiplications and additions among all 160 received samples. Time of calculation is significantly saved here. In second part of device, there are several parts which are executing their function one after another in a row. Every part uses some very complex elements which are same or similar, so it can be shared between them since those parts do not need them at same time. There is made significant reduction of resource usages.

7. References

[1] Timothy M. Schmidl and Donald C. Cox: Robust Frequency and Timing Synchronization for OFDM, December 1997

[2] H. Minn, M. Zeng, and V. K. Bhargava: On Timing Offset Estimation for OFDM Systems, July 2000

[3] Guangliang Ren, Yilin Chang, Hui Zhang, and Huining Zhang: Synchronization method based on a new constant envelop preamble for OFDM systems, March 2005

[4] Meng-Han Hsieh, Che-Ho Wei: A Low-Complexity Frame Synchronization and Frequency Offset Compensation Scheme for OFDM Systems over Fading Channels, September 1999

[5] M. Speth, S. A. Fechtel, G. Fock, H. Meyr: Optimum Receiver Design for Wireless Broad-Band Systems Using OFDM—Part I

[6] F. Harris: Orthogonal Frequency Division Multiplexing, January 2010

[7] 802.11a White paper, VOCAL Technologies, Ltd.