Area and Power Conscious Rake Receiver Design for Third Generation WCDMA Systems

Jina Kim

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Electrical Engineering

Dr. Dong S. Ha, Chairman
Dr. James. R. Armstrong
Dr. Joseph G. Tront

January 2003
Blacksburg VA

Keywords: WCDMA, Rake receiver, Multipath, Power dissipation, Circuit complexity, Bit truncation

Area and Power Conscious Rake Receiver Design for Third Generation WCDMA Systems

Jina Kim

Dr. Dong S. Ha, Chairman

Bradley Department of Electrical and Computer Engineering

(Abstract)

A rake receiver, which resolves multipath signals corrupted by a fading channel, is the most complex and power consuming block of a modem chip. Therefore, it is essential to design a rake receiver be efficient in hardware and power. We investigated a design of a rake receiver for the WCDMA (Wideband Code Division Multiple Access) system, which is a third generation wireless communication system. Our rake receiver design is targeted for mobile units, in which low-power consumption is highly important. We made judicious judgments throughout our design process to reduce the overall circuit complexity by trading with the performance. The reduction of the circuit complexity results in low power dissipation for our rake receiver. As the first step in the design of a rake receiver, we generated a software prototype in MATLAB. The prototype included a transmitter and a multipath Rayleigh fading channel, as well as a rake receiver with four fingers. Using the software prototype, we verified the functionality of all blocks of our rake receiver, estimated the performance in terms of bit error rate, and investigated trade-offs between hardware complexity and performance. After the verification and design trade-offs were completed, we manually developed a rake receiver at the RT (Register Transfer) level in VHDL. We proposed and incorporated several schemes in the RT level design to enhance the performance of our rake receiver. As the final step, the RT level design was synthesized to gate level circuits targeting TSMC 0.18 μm CMOS technology under the supply voltage of 1.8 V. We estimated the performance of our rake receiver in area and power dissipation. Our experimental results indicate that the total power

dissipation for our rake receiver is 56 mW and the equivalent NAND2 circuit complexity is 983,482. We believe that the performance of our rake receiver is quite satisfactory.

Acknowledgements

I first want to thank Dr. Dong S. Ha for initiating an interesting research project and for providing great support and guidance for the research. I would also like to extend my appreciation to Dr. James R. Armstrong and to Dr. Joseph G. Tront for serving on my advisory committee. Next, I would like to express my appreciation to my fellow researchers in the Virginia Tech VLSI for Telecommunications (VTVT) Laboratory. I would like to thank Hyung-Jin Lee, Jos Sulistyo, Nate August, Kye Hun Lee, Venkat Srinivasan, and Woo Cheol Chung. Particularly, I give very special gratitude to Hyung-Jin Lee for immense support both in and out of the Lab. Discussion with him always gives me a lot of great ideas. I wish them all great success in their future academic and professional lives. Finally, most of all, I thank my parents for their unconditional love and support. Their love and support play a crucial role in everything I do.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In the past decades, mobile communication systems have proliferated explosively and offer a wide variety of communication services. Mobile communication systems have evolved from the first generation based on analog technology to the third generation (3G) based on digital communications. The first two generations of mobile communication systems are limited mostly to voice communications, while the 3G communication systems intend to provide high data rate services including multimedia and video [Dal01, Zie01].

The second-generation systems based on Code Division Multiple Access (CDMA) technology employ digital wireless technologies that allow multiple users to share the same radio spectrum without interfering with each other. The 3G systems employ high-speed versions of CDMA called Wideband CDMA (WCDMA) or CDMA2000. The WCDMA system was co-developed by NTT DoCoMo and Ericson and is compatible with the Global System for Mobile Communication (GSM). The WCDMA system uses an asynchronous network between the base stations [KTF03]. The user equipment (UE) has no knowledge of the relative time difference between base stations [Qua01]. The major specifications for the WCDMA system are as follows. The bandwidth for a WCDMA system is 5 MHz, and the chipping rate is 3.84 Mcps. The maximum data rate is 9510 Kbps for the uplink, because there is a maximum of six data channels, each of which has 960 Kbps, and one control channel that has 150 Kbps. The downlink has maximum data rate of 1920 Kbps. The QPSK (Quadrature Phase Shift Keying) modulation scheme is employed in both directions [3G213].

CDMA2000 is the other 3G standard that competes with WCDMA. It is an upgrade for Interim Standard-95 (IS-95). CDMA2000 uses a synchronous network relying on the Global Positioning System (GPS). There are two versions of CDMA2000 standards called CDMA2000 1X and CDMA2000 3X [Pej01]. 1X uses a chipping rate of 1.2288 Mcps and provides the maximum data rates of 153.6 Kbps for the uplink and 307.2 Kbps for the downlink. 3X uses three carriers in parallel for higher data rates,

which are maximum 614.4 Kbps for the uplink and 1036.8 Kbps for the downlink. The chipping rate for 3X is 3.6864 Mcps.

In this thesis, we consider the WCDMA system, as it is dominant in the market coverage. WCDMA, which is based on GSM, shares 65% of the global market, while CDMA2000 has limited growth of about 12% due to the minor occupation of IS-95 [Kuo01]. Three major components of a WCDMA modem are a cell searcher, a channel decoder, and a rake receiver. A cell searcher is responsible for obtaining the timing of the slot boundary and detecting the scrambling code group and the primary scrambling code [3G214]. In the first step, a searcher acquires slot synchronization to a cell, which is typically done with a single matched filter. In the second step, the frame synchronization is acquired and the code group of the cell found in the first step is identified. In the last step, the exact primary scrambling code used for the found cell is determined. Steps 2 and 3 are based on the correlation operation. A channel decoder performs a decoding operation for the channel coding, which is employed for reliable communications. Convolutional coding is used with relatively low data rates in WCDMA, and turbo coding is for high data rates. Turbo and Viterbi decoding techniques are usually employed for the channel decoding. The rake receiver performs the core operations for a WCDMA modem including despread of the received signal, resolution of multipaths, and fine-tuning of the timing. Among the three major components, a rake receiver is the most complex in hardware design and is the focus of this thesis research. We considered implementation of a rake receiver for the downlink, which is on the mobile side.

A rake receiver has multiple fingers to resolve multipath signals corrupted by a fading channel. Each finger has a channel estimator and a channel compensator to mitigate the phase rotation due to the fading channel. A rake receiver also includes a time tracker for fine-tuning of the timing and a combiner to combine outputs of the fingers. Our design started with a software prototype of a rake receiver. A transmitter and a channel as well as a rake receiver were implemented in MATLAB and simulated to verify the functionality of internal blocks of the rake receiver and to measure the Bit Error Rate (BER). Finally, we implemented the rake receiver at the RTL (Register Transfer Level) level in VHDL and synthesized the VHDL code to achieve a gate level

netlist. We estimated the performance of our rake receiver in terms of the circuit complexity and power dissipation in TSMC 0.18 μm CMOS technology. The total power dissipation for our rake receiver is 56 mW and the equivalent NAND2 circuit complexity is 983,482 under 1.8 V.

The organization of the thesis is as follows. In Chapter 2, we provide background information necessary to understand the WCDMA systems. Chapter 3 describes the channel model used for our simulations and the downlink rake receiver design. MATLAB simulation results for the verification of the function and the BER measurement of the rake receiver are also presented. Chapter 4 provides details of the hardware design of the rake receiver. Experimental results including power consumption and the area of the synthesized circuit are given in Chapter 5. Finally, Chapter 6 concludes our work on the rake receiver design.

Chapter 2 WCDMA Physical Layer Description

WCDMA uses a direct sequence spread spectrum technique, in which a narrow band signal is spread to a wide band signal. Figure 2.1 shows the narrow band signal before the spreading and the wide band signal after the spreading. The processing gain, which is defined as W/R, decreases with the increase of the bit rate. For the WCDMA systems, a variable data rate within a fixed bandwidth is achieved through the variable spreading and a high data rate through multicode transmission techniques. The spreading factor is reduced with variable spreading, when the data rate increases, while high bit rate services are mapped onto different parallel channels in multicode transmission [Gue01].

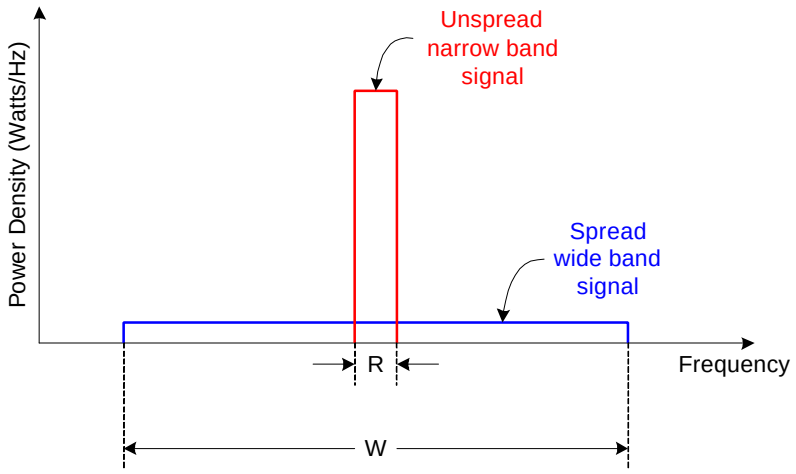


Figure 2.1: Processing gain and the spreading [Hol00]

In this section, we provide preliminaries of the WCDMA systems necessary to understand our rake receiver design. We describe Layer 1, which is also known as the physical layer, of the radio access network of the WCDMA systems operating in Frequency Division Duplex (FDD) mode. Since our receiver is designed for downlink, we focus on the downlink channel of the WCDMA physical layer.

Downlink is the communication channel from a base station to the mobile. There is only one dedicated physical channel for the downlink, which is called the Dedicated Physical Channel (DPCH). The Common Pilot Channel (CPICH) is one of common

downlink physical channels. Each channel is channelized with a channelization code, and is I/Q separated. The channelized complex chips are combined together and scrambled with a complex-valued scrambling code. Finally, the complex-valued chip sequence is modulated for transmission through the channel.

2.1 Physical channels

2.1.1 Dedicated physical channel

Unlike the uplink, there is only one dedicated physical channel for the downlink. Thus, the downlink DPCH is a time multiplex of a downlink Dedicated Physical Data Channel (DPDCH) and downlink Dedicated Physical Control Channel (DPCCH). The frame structure of DPCH is shown in Figure 2.2.

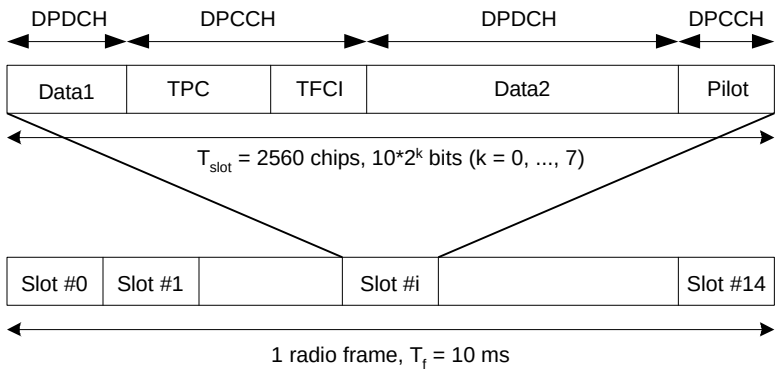


Figure 2.2: Frame Structure for DPCH

Each frame is 10 ms long and is split into 15 slots. Each slot has 2560 chips. The parameter k in Figure 2.2 determines the spreading factor SF as $SF = \frac{512}{2^k}$, and determines the total number of bits per DPCH slot. Since k ranges from 0 to 7, the spreading factor for DPCH is in the range of 4 to 512.

The downlink DPCH may or may not include a Transport Format Combination Indicator (TFCI). TFCI carries from 1 to 10 bits of transport format information in a slot. Transport format information is used to regulate the channel formation between the

physical layer and the Medium Access Control (MAC) layer. The sub-fields and their bits of the TFCI are defined in [3G211]. The Pilot bits are used for the channel estimation at the terminal for the dedicated channel and for providing the channel estimation reference for the common channels when they are not associated with the dedicated channels or not involved in the adaptive antenna techniques in the same way CPICH is. The mobile receiver can utilize CPICH and/or the Pilot bits within DPCCH for the same purposes. Transmit Power Control (TPC) is used for adjusting the mobile transmit power in order to keep the signal-to-interference ratio (SIR) at a given SIR target. The number of bits and the patterns for the Pilot and TPC are also defined in [3G211]. Since we do not use the information of the DPCCH of DPCH for our rake receiver, detailed information on TFCI, Pilot, and TPC is omitted.

2.1.2 Common pilot channel

The CPICH, which is shared by all users of a base station, carries a pre-defined bit sequence with a fixed rate (30 kbps, SF = 256). Unlike the uplink, where the pilot symbols are included in the DPCCH with other control information, the pilot symbols are transmitted through CPICH in downlink. Hence, CPICH is used for the estimation of the channel phase rotation and the frequency offset. The frame structure of CPICH is shown in Figure 2.3. When transmit diversity is not employed, the pre-defined bit sequence is all 1's that is, in fact, the binary bit 0.

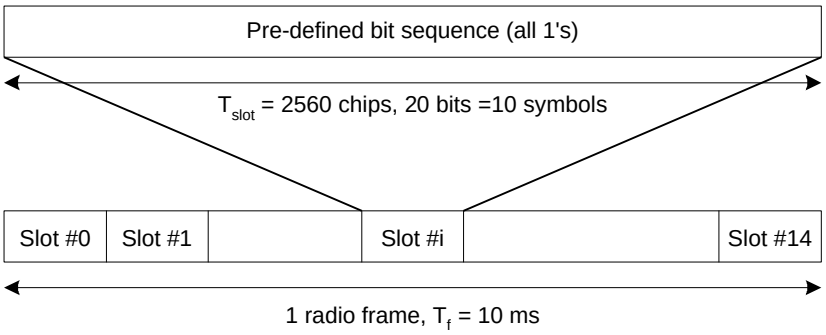


Figure 2.3: Frame Structure of CPICH

2.2 Spreading

The spreading operation consists of channelization and scrambling operations. Figure 2.4 shows the block diagram of the spreading operation. Each pair of two consecutive symbols in a channel is converted to parallel and is applied to both I and Q branches. The I and Q branches are channelized by multiplying the same Orthogonal Variable Spreading Factor (OVSF) code, which is a channel specific. Then, DPCH and CPICH are scrambled by the complex-valued scrambling codes, and combined.

In Figure 2.4, C_d and C_p denote the OVSF code for DPCH and CPICH, respectively. S_d and S_p are the scrambling code for DPCH and CPICH. DPCH and CPICH are channelized with different OVSF codes with necessary spreading factors, and then scrambled also with different complex-valued scrambling codes specific to a cell.

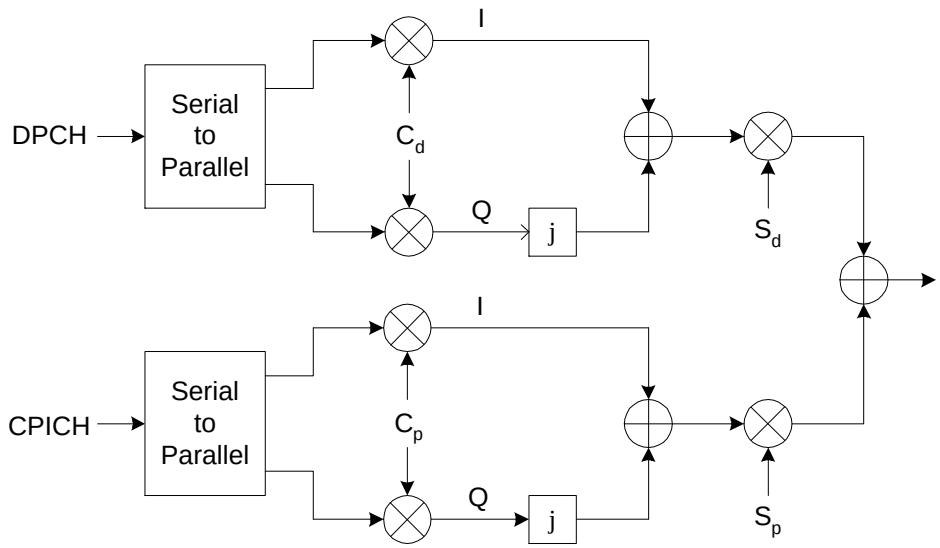


Figure 2.4: Spreading of DPCH/CPICH

2.2.1 Channelization

The channelization operation for the downlink separates the downlink connections to different users within one cell. The channelization is performed by multiplying appropriate OVSF codes to the I and Q branches. Through the channelization, a symbol is spread into a number of chips according to the spreading

factor, which results in the increase of the bandwidth. By using the different spreading factors for each channel, variable data rates can be obtained. Since the spread signal bandwidth is the same for all users, multiple spreading factors are needed for the multiple rate transmission.

Channelization codes are picked from the code generation tree shown in Figure 2.5, which is based on the OVSF technique. Each stage has two sub branches. The upper branch is obtained by repeating the code of the former stage and the lower branch is obtained as the concatenation of the original and the reversed codes of the former stage. The process is summarized in Figure 2.6.

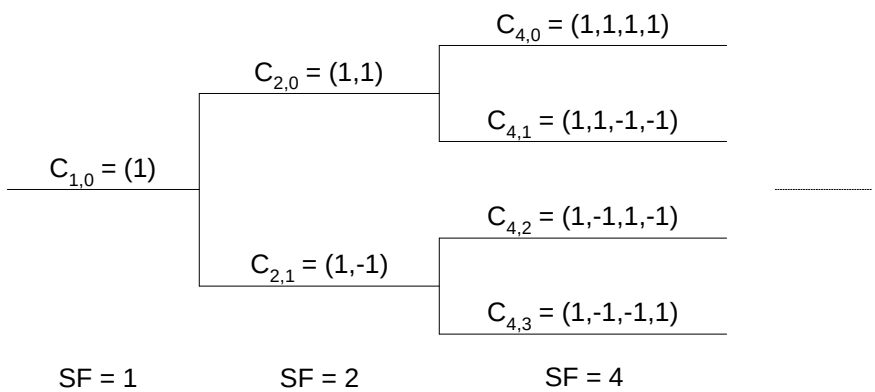


Figure 2.5: OVSF Code Tree

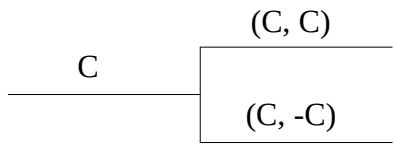
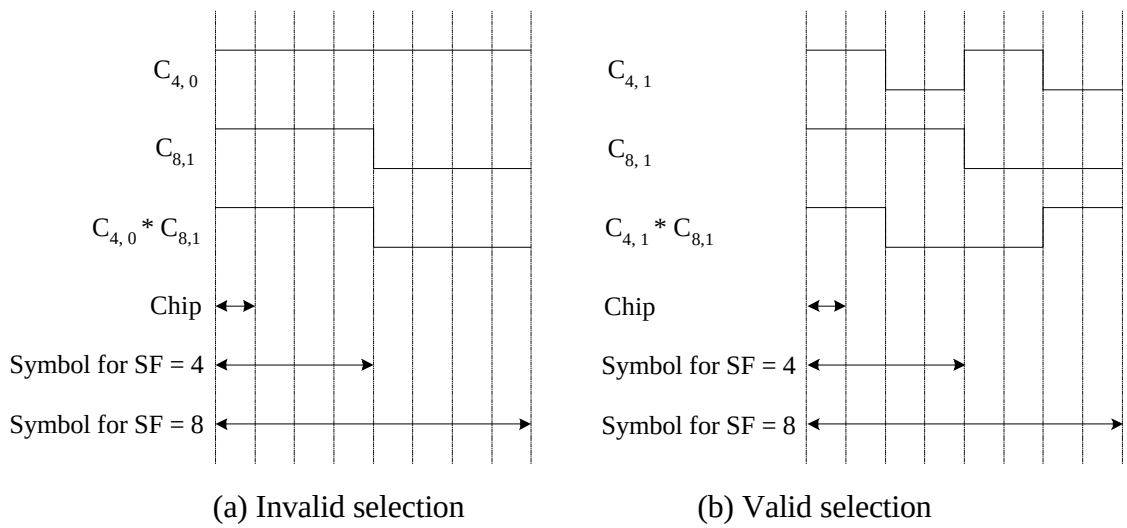


Figure 2.6: One Stage of the Tree Structure

OVSF codes should be selected to maintain the orthogonality between different downlink connections. When two channels having the same transmission rate are spread, they should pick two different OVSF codes with the desired spreading factor. In case two channels use different transmission rates, the OVSF code for the lower transmission rate should not be the child of the OVSF code for the higher transmission rate. Figure 2.7

illustrates an example selection of OVSF codes. In case of Figure 2.7(a), $C_{4,0}$ and $C_{8,1}$ are selected as OVSF codes for the channels, and $C_{8,1}$ is a child of $C_{4,0}$. The correlation value of the two OVSF codes during the symbol period for SF = 4 is four, though that of SF = 8 is zero. Hence, the selection is invalid. Case (b) selects $C_{4,1}$ and $C_{8,1}$, and note $C_{8,1}$ is not a child of $C_{4,1}$. The correlation of the two OVSF codes is zero for both SF = 4 and SF = 8, which is a valid selection.



(a) Invalid selection (b) Valid selection

Figure 2.7: An example of the OVSF code selection for orthogonality

Figures 2.8 and 2.9 illustrate the correlation values between different OVSF codes. Figure 2.8 shows the correlation values between $C_{256,100}$ and $C_{256,i}$ ($i = 0,1,\cdots 255$). The auto-correlation value of $C_{256,100}$ itself is 256, while all cross-correlation values with $C_{256,100}$ are zero, which means that the two different OVSF codes are orthogonal. Figure 2.9 shows correlation values between OVSF codes $C_{4,0}$ and its children and non-children OVSF codes. As can be seen from the figure, there is no orthogonality between the OVSF code $C_{4,0}$ and its children codes, while non-children codes are all orthogonal to it, independent of the spreading factor.

Since CPICH is always spread by $C_{256,0}$, the OVSF codes for DPCH should be chosen as $C_{SF,k}$ where SF is the spreading factor and $SF/4 \le k \le SF$. Then, it is guaranteed that the OVSF code for DPCH is not the parent of the OVSF code for DPCH, $C_{256,0}$. So the two channels are orthogonal.

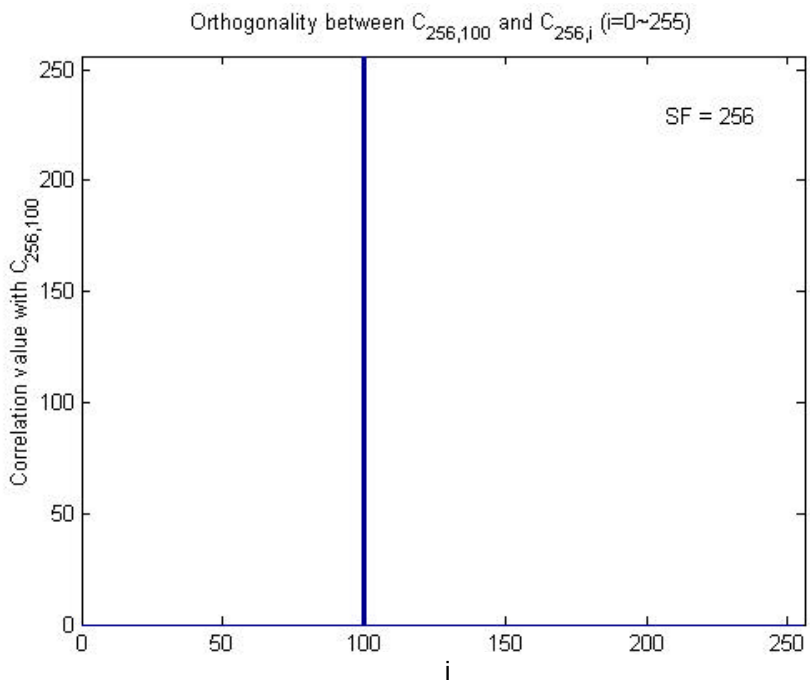


Figure 2.8: Correlation Between OVSF Codes Having the Same Spreading Factor 256

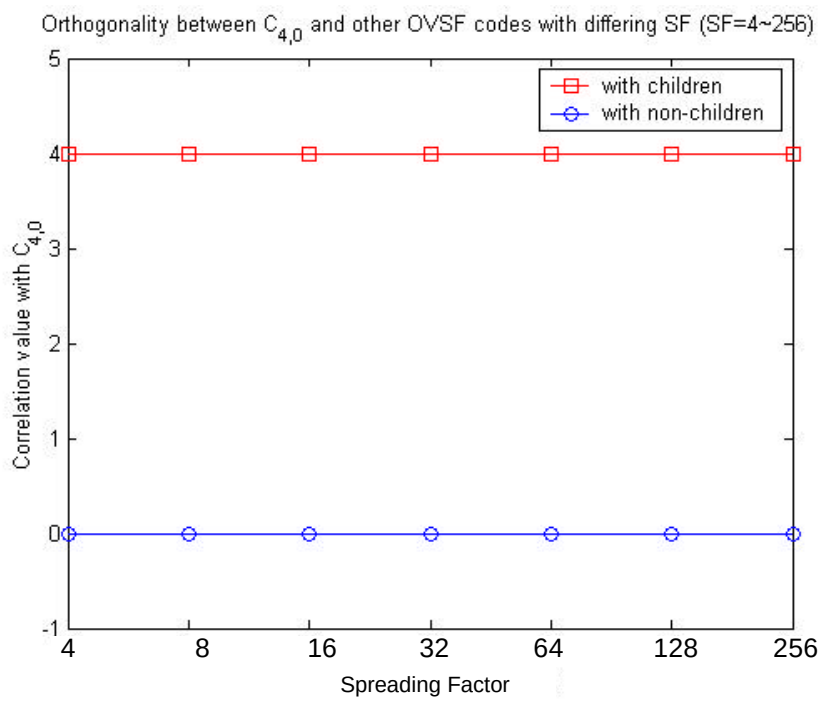


Figure 2.9: Correlation Between $C_{4,0}$ and other OVSF Codes Having Different Spreading Factors

2.2.2 Scrambling

The scrambling operation is applied to discriminate different cells, and the scrambling codes are assigned by higher layers. The spread complex sequences of DPCH and CPICH are summed together after the scrambling operation as shown in Figure 2.4.

A total of 2^{18}-1 (= 262,143) scrambling codes can be generated. However, not all the scrambling codes are used. The scrambling codes are divided into 512 sets each of a primary scrambling code and each set has 15 secondary scrambling codes. Hence, 8192 scrambling codes are practically used. The set of primary scrambling codes is further divided into 64 scrambling code groups, each consisting of 8 primary scrambling codes. Each cell is allocated one and only one primary scrambling code. The CPICH are always transmitted using the primary scrambling code, and DPCH can be transmitted with either the primary scrambling code or a secondary scrambling code from the set associated with the primary scrambling code of the cell [3G213].

The scrambling code sequences are constructed by combining two real sequences into complex sequences, and these two real sequences are Gold sequences generated from two binary m-sequences. The two m-sequences, which are labeled as x and y in Figure 2.10, are formed by 18-tap linear feedback shift registers (LFSR's). The x sequence is generated by the primitive polynomial $1+X^7+X^{18}$ and the y sequence by $1+X^5+X^7+X^{10}+X^{18}$. The initial conditions for the x sequence and y sequence are independent of the scrambling code number n. The initial condition of the LFSR for the x sequence is $x(0)=1, x(1)=x(2)=\ldots =x(17)=0$, and that for the y sequence is all 1.

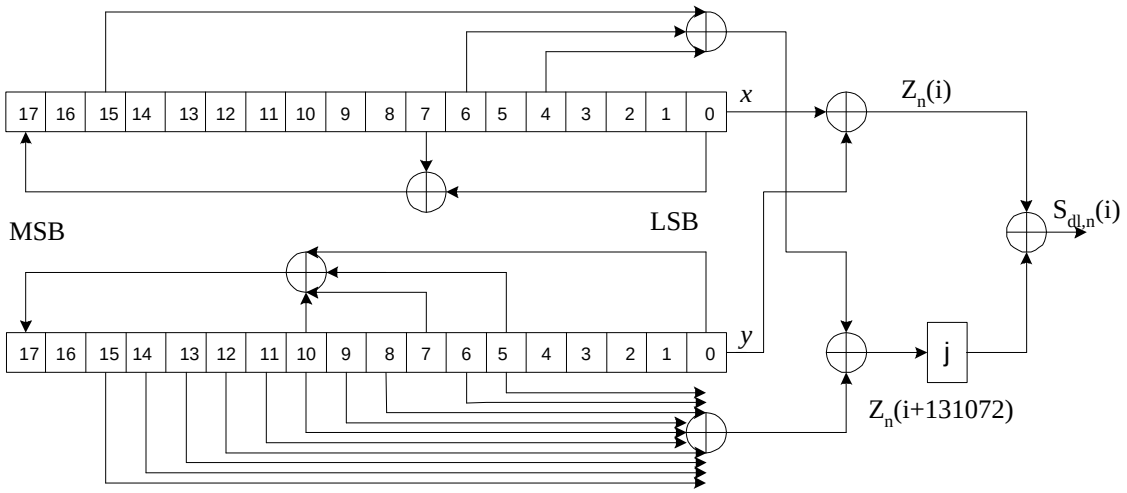


Figure 2.10: Scrambling Code Generator

The cross-correlation property of Gold sequences is known to be better than true random sequences [Din98]. The scrambling codes provide not only the low cross-correlation property, but also low auto-correlation values between the two time-shifted versions, so that multipaths with different time delays can be resolved. Figure 2.11 shows the auto-correlation property between two time-shifted versions of a long scrambling code with the spreading factor of 256. The auto-correlation value of the scrambling code without time-shift is 256 in the figure, while it is nearly zero with any time-shift. The WCDMA system uses complex valued scrambling codes, which are the real and imaginary combinations of two Gold sequences, to exploit these correlation characteristics.

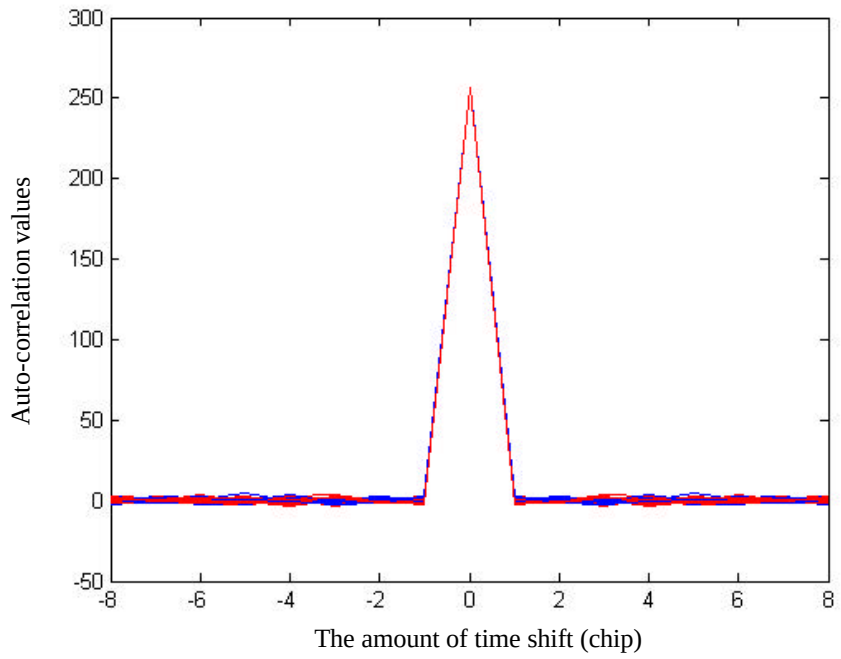


Figure 2.11: Auto-correlation property of the scrambling codes

2.3 Modulation

The complex valued chip sequence after the scrambling operation is QPSK modulated as shown in Figure 2.12.

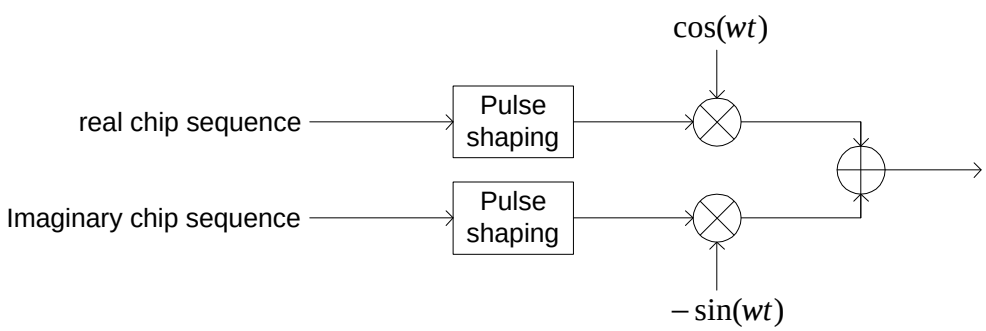


Figure 2.12: Modulation

The pulse shaping filter is employed in order to remove intersymbol interference (ISI). Thus, it is desirable to use a pulse shape whose spectrum drops sharply at the adjacent

channels [Raz98]. For example, a raised-cosine signal provides a tight specturm and ISI-free operation. However, in practical systems like WCDMA, the raised-cosine filter is decomposed into two sections, one placed in the transmitter side and the other in the receiver side, so that the receiver operates as a matched filter. Figure 2.13 shows the decomposition of a raised-cosine filter into two root-raised cosine filters.

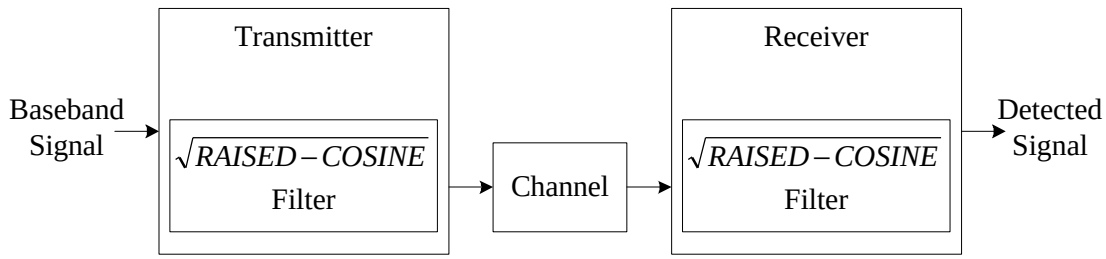


Figure 2.13: Decomposition of a Raised-Cosine Filter into Two Sections

In WCDMA, a Root-Raised Cosine (RRC) filter with a roll-off factor of α = 0.22 is employed for pulse shaping. The impulse response of the RRC filter is given as

$$RC_0(t) = \frac{\sin\left(p\,\frac{t}{T_C}(1-a)\right) + 4a\,\frac{t}{T_C}\cos\left(p\,\frac{t}{T_C}(1+a)\right)}{p\,\frac{t}{T_C}\left(1-\left(4a\,\frac{t}{T_C}\right)^2\right)}$$

where $T_c = \frac{1}{chiprate} = \frac{1}{3.84 Mchips/\sec} \approx 0.26042 ms/chip$ is the duration of a chip and α= 0.22 [3G101]. The impulse response of the pulse shaping filter is shown in Figure 2.14 for the one chip duration, from $-T_c/2$ to $T_c/2$.

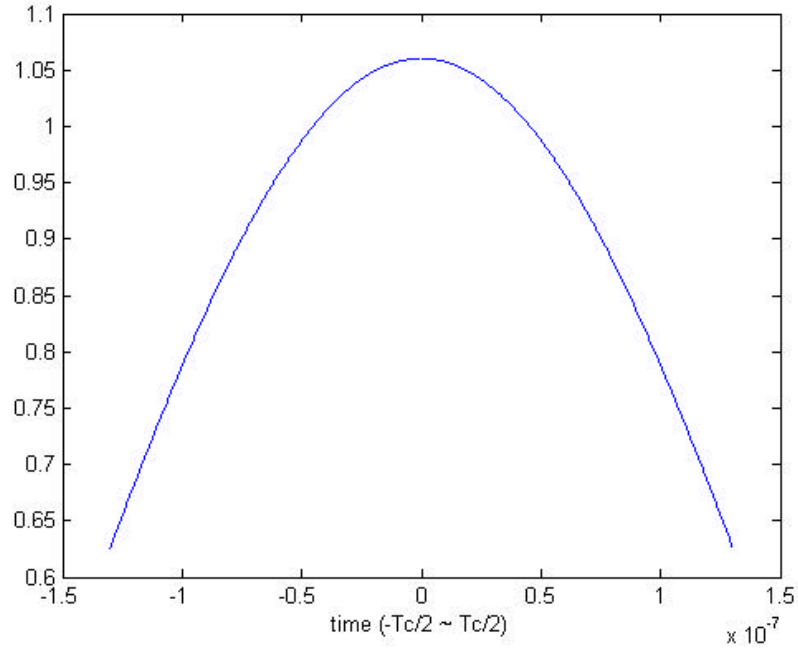


Figure 2.14: Impulse Response of the Transmit Pulse-Shaping Filter

2.4 Concluding remark

In this chapter, we described the physical layer of WCDMA briefly, and detailed the spreading and modulation. In the subsequent chapters we present the design of a rake receiver based on the preliminary knowledge covered in this chapter.

Chapter 3 Channel and Simulation Model

This chapter describes a channel model and simulation models of our rake receiver for function verification and performance evaluation.

3.1 Channel Model

Transmitted signals propagated through a radio channel are affected by fading. Two major physical factors that influence the fading are multipath propagation and the speed of the mobile. The presence of reflecting objects and scatters in the channel creates a constantly changing environment that dissipates the signal energy in amplitude, phase, and time. The relative motion between the base station and the mobile results in a frequency modulation due to different Doppler shifts on each of the multipath components [Rap96]. At the receiver front end, the additive white Gaussian noise (AWGN) is introduced by the thermal noise of the electric components of the receiver.

Figure 3.1 shows the channel model with four multipaths, which is used for simulation of our rake receiver. The transmitted signal propagates through four different time variant channels and produces four multipaths. The rake receiver receives the multipath signals added with AWGN.

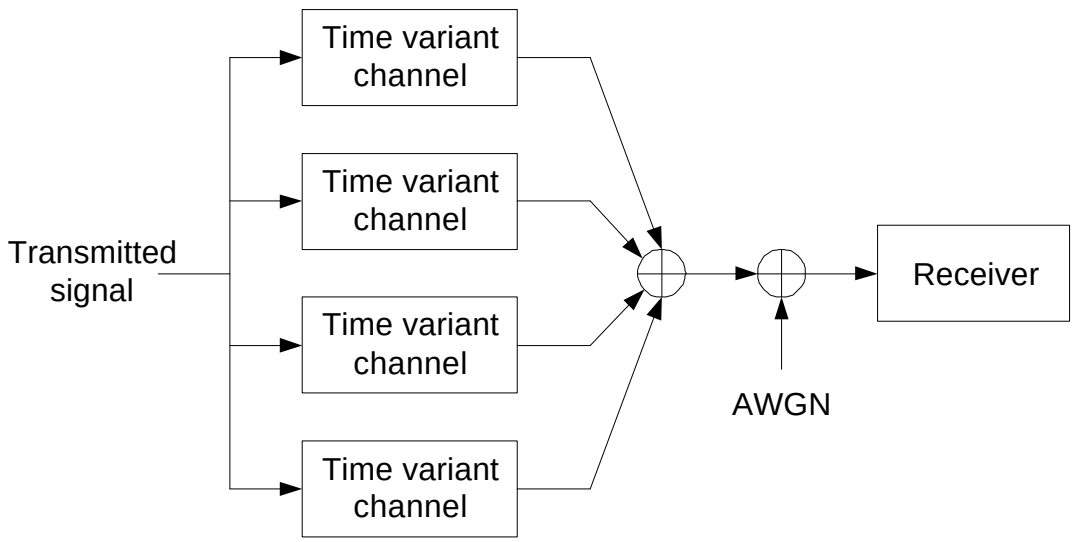


Figure 3.1: Channel model for the simulation

3.1.1 Time-Variant Multipath Environment

As noted, two major factors influence the fading of the radio propagation channel. One is the multipath propagation, and the other the mobile speed. A multipath signal propagation through a radio channel is a time-variant channel impulse response and can be expressed as in (3.1).

$$c(t;t) = \sum_{k=1}^{L} a_k(t) d(t - t_k) \tag{3.1}$$

where $\{a_k(t)\}$ represents a time-variant attenuation factor for the kth multipath propagation path and $\{\tau_k\}$ is the corresponding time delay [Rap96]. As can be seen in (3.1), the time-variant channel characteristics cause a constantly changing amplitude and a phase to the transmitted signals, and these characteristics can be statistically modeled with the Rayleigh distribution. The Rayleigh distribution can be obtained by calculating the sum of two quadrature Gaussian noise signals.

The second factor affecting fading is the mobile speed. The relative motion between the base station and the mobile results in frequency modulation due to a Doppler shift on each of the multipath components. Let the angle between the base station and the moving direction of the mobile be θ. Then, the Doppler shift f_d can be expressed as

$$f_d = \frac{v}{l} \cdot \cos q \tag{3.2}$$

where v is the constant velocity of the moving mobile and λ is the wavelength. When the angle θ is zero, Doppler shift has the maximum value, which is f_m $(= v/l)$.

Therefore, to incorporate a simulation environment that contains multipath fading and Doppler shifts, Clarke's fading model is employed in our simulation [Smi75]. As shown in Figure 3.2, two independent Gaussian random variables form in-phase and quadrature-phase fading branches in the frequency domain. Random signals in the frequency domain are shaped into time domain waveforms with Doppler fading through a Doppler power spectral filter and the inverse fast Fourier transform (IFFT). Then, a complex valued Rayleigh fading signal with the proper Doppler spread is obtained at the output of the channel.

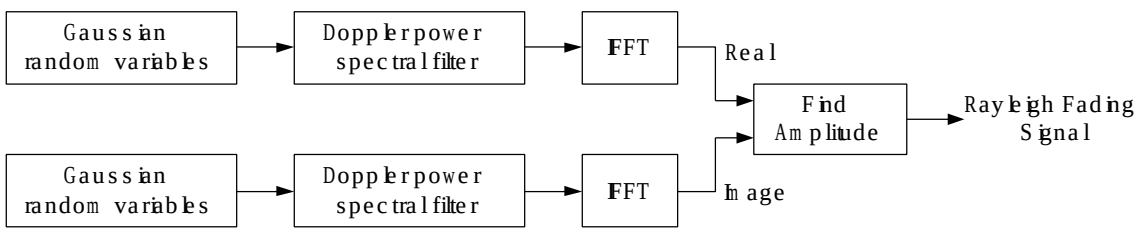


Figure 3.2: Frequency domain implementation of a Rayleigh fading channel [Rap96]

Rayleigh fading signals with two different mobile speeds, 5 Km/h and 120 Km/h, are shown in Figure 3.3 and Figure 3.4, respectively. The mobile speed of 5 Km/h and 120 Km/h correspond to the Doppler frequency of 9 Hz and 222 Hz, respectively, and the x-axis is the period of 50 ms, which corresponds to 5 time frames. The Rayleigh fading channel, which is realized by multiplying the value of a complex valued Rayleigh fading signal by the value of a transmitted signal, causes a time-varying phase rotation and an amplitude distortion of the transmitted signal.

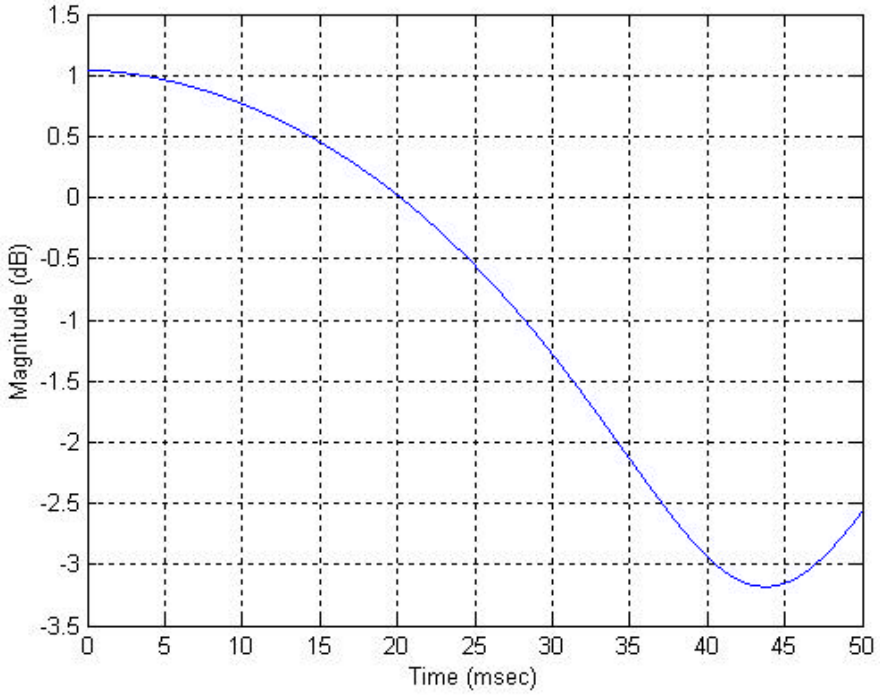


Figure 3.3: Rayleigh fading signal with the mobile speed of 5 Km/h

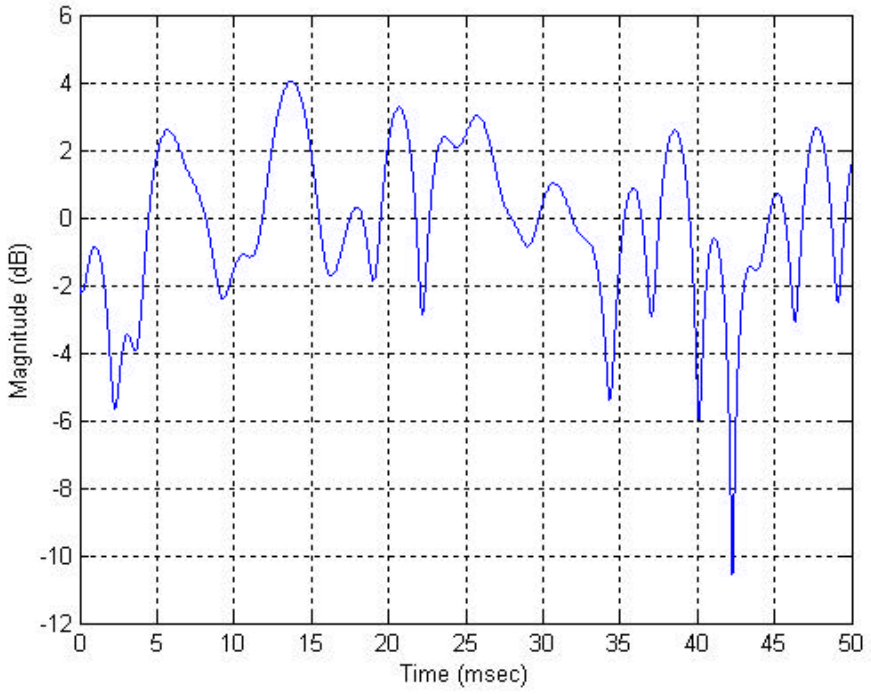


Figure 3.4: Rayleigh fading signal with the mobile speed of 120 Km/h

3.1.2 Additive White Gaussian Noise

Additive White Gaussian Noise (AWGN) is added to the front end of the receiver. The signal to noise ratio (SNR) of the received signal varies from 0 dB to 12 dB in our simulation. The variance of the AWGN, σ, is fixed to one, thus the noise energy N_0, that is $2\sigma^2$, has the value of 2. Since the noise energy is fixed, the signal energy per bit, E_b varies according to the SNR. In the simulation, the AWGN is added to the signal that passes through the Rayleigh fading channel. The signal energy per bit E_b and per chip E_c is obtained as follows for a given SNR in dB assuming without pulse shaping.

$$\begin{aligned} SNR\_dB &\equiv 10\log SNR \\ SNR &= 10^{SNR\_dB/10} \end{aligned} \tag{3.3}$$

Noting

$$SNR = \frac{E_b}{N_0} \tag{3.4}$$

E_b and E_c are obtained as

$$E_b = N_0 \cdot SNR = \left(2s^{\ 2}\right)\cdot\left(10^{SNR\_dB/10}\right) = 2\cdot\left(10^{SNR\_dB/10}\right) \tag{3.5}$$

$$E_c = E_b / SF = \frac{2\cdot\left(10^{SNR\_dB/10}\right)}{SF} \tag{3.6}$$

where SF is the spreading factor.

3.2 Rake Receiver Design

The receiver design should aim the recovery of data from a multipath environment. The most commonly adopted receiver architecture a multipath environment is a rake receiver shown in Figure 3.5.

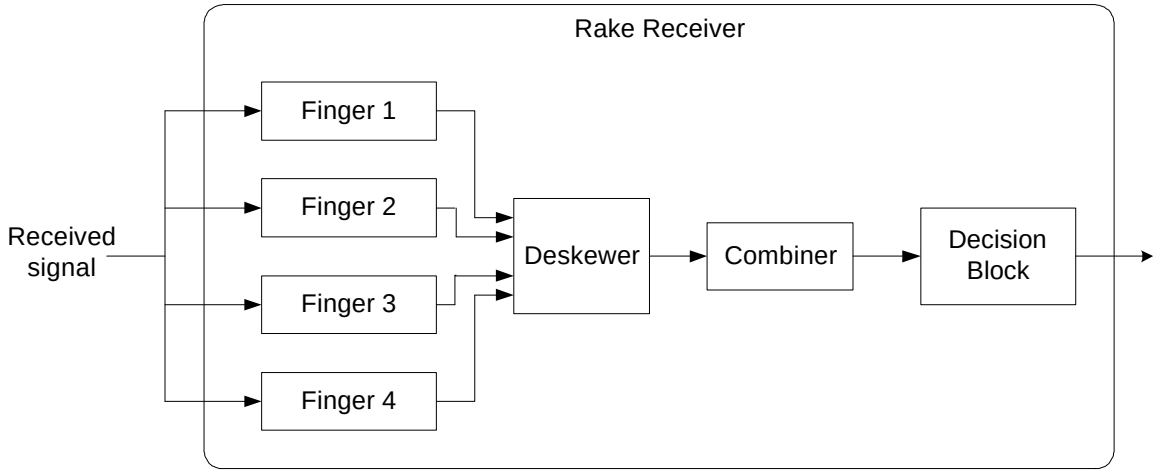


Figure 3.5: Rake receiver structure

A rake receiver has multiple fingers to resolve multipaths that have different path delays. A finger receives a signal containing all the multiple paths then processes one of the multipaths assigned to it. Each multipath has a different distortion that is mainly caused by the Rayleigh fading, AWGN, and the frequency offset between the transmitter and the receiver oscillators. Therefore, the received signal of a finger should undergo phase compensation and frequency offset compensation specific to the multipath as well as the despreading process. A power estimator measures the signal power of the multipath to determine if it is strong enough to be considered further. The deskewer compensates individual multipath delays. After these processes, the multipath path signals are combined to recover the desired data.

In our research, we consider a rake receiver with four fingers, so that up to four multipaths can be processed. The process is explained in the following sections.

3.2.1 Received Signal

The transmitted signal for the downlink is expressed in (3.7), where D_i and D_q are the I and Q channel symbols of DPCH, respectively, and P_i and P_q are the I and Q channel symbols of CPICH, respectively. C_d is the OVSF code of DPCH, and C_p is the OVSF code of CPICH. The complex valued scrambling code of DPCH is denoted as $S_{di} + j \cdot S_{dq}$ and $S_{pi} + j \cdot S_{pq}$ is the scrambling code of CPICH.

$$
\begin{aligned}
&T_i + j \cdot T_q \\
&= (D_i \cdot C_d + j \cdot D_q \cdot C_d) \cdot (S_{di} + j \cdot S_{dq}) + (P_i \cdot C_p + j \cdot P_q \cdot C_p) \cdot (S_{pi} + j \cdot S_{pq}) \\
&= \{(D_i \cdot C_d \cdot S_{di} - D_q \cdot C_d \cdot S_{dq}) + j \cdot (D_i \cdot C_d \cdot S_{dq} + D_q \cdot C_d \cdot S_{di})\} \\
&+ \{(P_i \cdot C_p \cdot S_{pi} - P_q \cdot C_p \cdot S_{pq}) + j \cdot (P_i \cdot C_p \cdot S_{pq} + P_q \cdot C_p \cdot S_{pi})\}
\end{aligned} \tag{3.7}
$$

Then, the received signal through the channel can be expressed as

$$
R_i + j \cdot R_q = (T_i + j \cdot T_q) \cdot \{a \cdot \exp(jq)\} \cdot \{\exp(jf)\} + h_0 . \tag{3.8}
$$

In (3.8), α and θ denote the gain and the phase rotation of the channel due to the Rayleigh fading, and f is the frequency offset between the transmitter and the receiver. h_0 represents the AWGN at the receiver front end. To remove the performance degradation due to the effects of the Rayleigh fading and the frequency offset, they should be estimated and compensated.

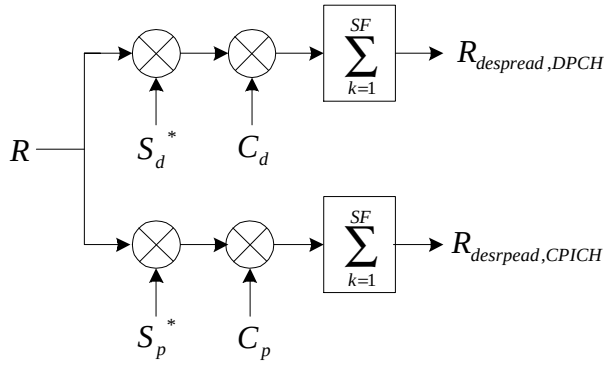
3.2.2 Despreading

The despreading operation converts the rate of the received signal from the chipping rate to the symbol rate of a received signal. For despreading, the complex conjugations of the scrambling codes for DPCH and CPICH are multiplied to the received signal followed by the OVSF codes. (3.9) and (3.10) indicate the result of the despreading operation. In (3.9) and (3.10), $\sum_{k=1}^{SF} C_i C_i = \sum_{k=1}^{SF} C_q C_q \cong 1$, $\sum_{k=1}^{SF} C_i C_q \cong 0$ are assumed. We ignore the coding gain SF, which is the spreading factor for each channel.

$$
R_{desrpead,DPCH} = \sum_{k=1}^{SF} \left[R \cdot (S_{di} - jS_{dq}) \cdot C_d \right] = 2(D_i + jD_q) \cdot \{a \exp(jq) \cdot \exp(jf)\} + h_0 \tag{3.9}
$$

$$
R_{desrpead,CPICH} = \sum_{k=1}^{SF} \left[R \cdot (S_{pi} - jS_{pq}) \cdot C_p \right] = 2(P_i + jP_q) \cdot \{a \exp(jq) \cdot \exp(jf)\} + h_0 \tag{3.10}
$$

Figure 3.6 shows the block diagram of the despreading process. The despread DPCH and CPICH symbols still contain the effect of AWGN, the Rayleigh fading channel and the frequency offset. In the following sections we discuss the compensation for the channel phase rotation and the frequency offset.



Note: S^* is the complex conjugate of S

Figure 3.6: Despreading block

3.2.3 Phase rotation

Since the effect of the Rayleigh fading is dominant on the transmitted signal, a phase rotation due to the Rayleigh fading channel is estimated first under the assumption that a frequency offset does not exist. Figure 3.7 shows the performance of a receiver when the channel is subject to AWGN and the Rayleigh fading, but the receiver does not have the channel estimator and channel compensator. As shown in the figure, when the transmitted signal is affected only by AWGN, the rake receiver performance is acceptable. However, when the transmitted signal passes through the Rayleigh fading channel, the performance degradation is severe. The receiver fails to recover the desired data.

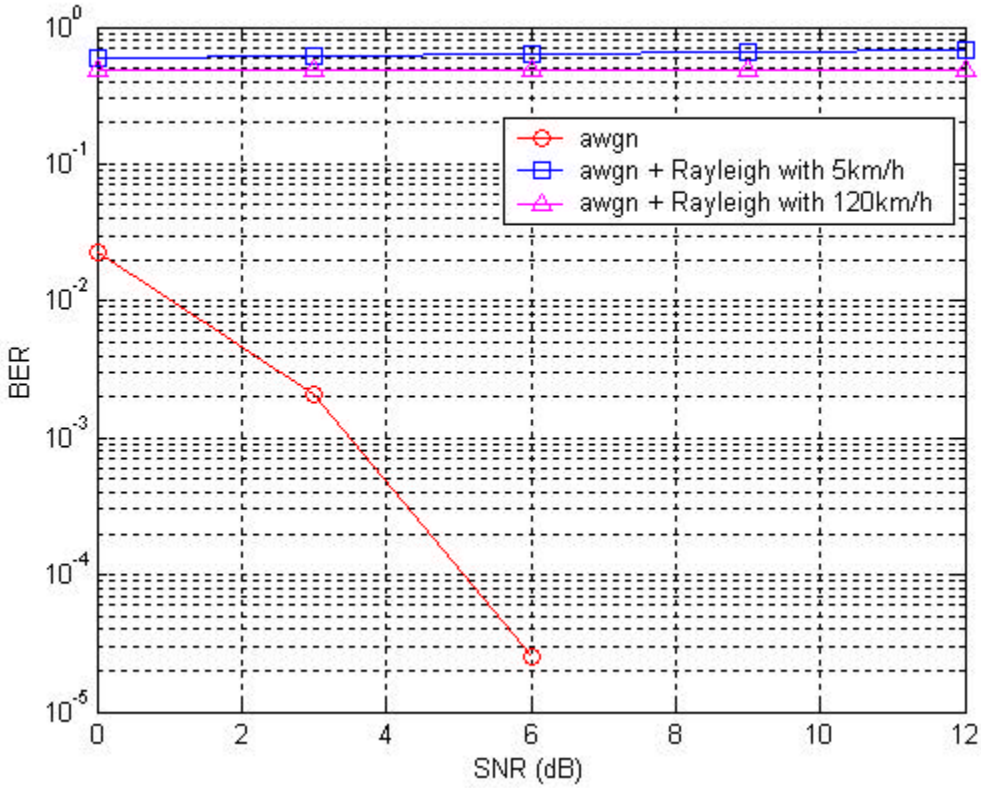


Figure 3.7: Performance without channel estimator and channel compensator

To eliminate the effect of the phase rotation, which is specified as $\exp(jq)$ in (3.9), and to receive the desired data symbols D_i and D_q, the complex conjugation of the phase rotation value, which is $\exp(-jq)$, should be multiplied by (3.9). The phase rotation of the channel can be estimated from CPICH, because the symbol values of CPICH are already known. As described in Section 2.1.1.2 and specified in [3G211], CPICH symbols P_i and P_q always have the value of 1's.

If the AWGN and the frequency offset are ignored, the despread CPICH, which is given in (3.10), can be rewritten as in (3.11).

$$\hat{R}_{despread,CPICH} = 2(P_i + jP_q) \cdot a \exp(jq) \tag{3.11}$$

Thus, by simply multiplying $(P_i - j \cdot P_q)$, which is *(1-j)*, the phase rotation value, $2a \exp(jq)$, is obtained. The detailed process for the estimation of the channel phase rotation is provided in Section 3.2.4.

3.2.4 Channel Estimation

We suggest two different methods for the channel estimation, both explained below.

Method 1: Use the received signal before the dispreading operation.

The pilot part of the received signal without the frequency offset (i.e. f = 0) is obtained from (3.8) and is given in (3.12).

$$\hat{R}_i + j \cdot \hat{R}_q = (T_i + j \cdot T_q) \cdot \{a \cdot \exp(jq)\} + h_0 \tag{3.12}$$

Since the DPCH and CPICH use different channelization codes, the data symbols D_i and D_q can be removed after applying the CPICH channelization code C_p from (3.7). Also, as the pilot signal values P_i and P_q are all 1's, the received signal for the channel estimation can be rewritten as in (3.13).

$$\begin{aligned}
&\hat{R}_i + j \cdot \hat{R}_q \\
&= \{(C_p \cdot S_{pi} - C_p \cdot S_{pq}) + j \cdot (C_p \cdot S_{pq} + C_p \cdot S_{pi})\} \cdot \{a \cdot \exp(jq)\} + h_0 \\
&= [(C_p S_{pi} - C_p S_{pq}) \cdot a \cos(q) - (C_p S_{pq} + C_p S_{pi}) \cdot a \sin(q)] \\
&\quad + j[(C_p S_{pi} - C_p S_{pq}) \cdot a \sin(q) + (C_p S_{pq} + C_p S_{pi}) \cdot a \cos(q)] + h_0
\end{aligned} \tag{3.13}$$

In order to find the *cos(q)* and *-sin(q)* in (3.12), we multiply $\hat{R}_i$ by the CPICH scrambling codes S_{pi} and S_{pq}, respectively. We obtain

$$\hat{R}_i \cdot S_{pi} = C_p \cdot a \cos(q) - C_p \cdot a \sin(q) + h_0 \tag{3.14}$$

and

$$\hat{R}_i \cdot S_{pq} = -C_p \cdot a \cos(q) - C_p \cdot a \sin(q) + h_0 \tag{3.15}$$

After despreading each of them and ignoring the coding gain of 256, we have

$$\sum_{k=1}^{256} \left(\hat{R}_i \cdot S_{pi}\right) \cdot C_p = a \cos(q) - a \sin(q) + h_0 \tag{3.16}$$

and

$$\sum_{k=1}^{256} \left(\hat{R}_i \cdot S_{pq}\right) \cdot C_p = -a \cos(q) - a \sin(q) + h_0 \tag{3.17}$$

Using (3.16) and (3.17), the channel estimation values *cos(q)* and *-sin(q)* can be written as

$$\sum_{k=1}^{256}\left(\hat{R}_i \cdot S_{pi}\right)\cdot C_p + \sum_{k=1}^{256}\left(\hat{R}_i \cdot S_{pq}\right)\cdot C_p = -2\cdot a\,\sin(q) + h_0 \tag{3.18}$$

and

$$\sum_{k=1}^{256}\left(\hat{R}_i \cdot S_{pi}\right)\cdot C_p - \sum_{k=1}^{256}\left(\hat{R}_i \cdot S_{pq}\right)\cdot C_p = 2\cdot a\,\cos(q) + h_0 \tag{3.19}$$

The gain or path loss term a is dependent on the path loss of each multipath. The path loss a should be maintained through the subsequent processes for the case of maximal ratio combining. The constant gain term 2 is also considered to maximize the coding gain, which is a shift right 1-bit operation in hardware. Figure 3.8 shows the structure of channel estimation method 1 without a moving average (which is necessary to account for the AWGN).

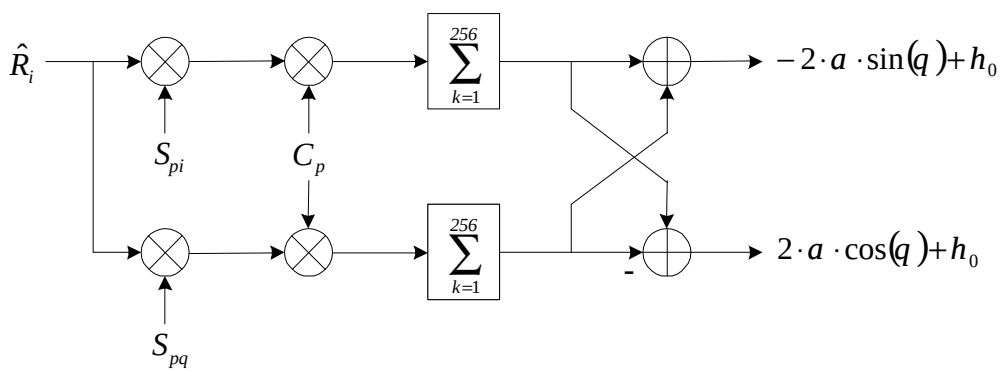


Figure 3.8: Channel estimation method 1 without a moving average

Method 2: Use the despread pilot symbols

The second method uses the despread pilot symbols under the same assumption that the frequency offset does not exist. The despread CPICH symbol shown in (3.10) can be rewritten as

$$\begin{aligned}
&\hat{R}_{desrpead,CPICH} \\
&= 2(P_i + jP_q)\cdot\{a\,\exp(jq)\} + h_0 \\
&= 2(1+j)\cdot[a\cdot\cos(q) + ja\cdot\sin(q)] + h_0 \\
&= 2\cdot a\cdot[\{\cos(q)-\sin(q)\} + j\{\cos(q)+\sin(q)\}] + h_0
\end{aligned} \tag{3.20}$$

The real and imaginary part of the despread CPICH components are

$$\mathrm{Re}\{R_{despread,CPICH}\} = 2 \cdot a \cdot [\cos(q) - \sin(q)] + h_0 \tag{3.21}$$

$$\mathrm{Im}\{R_{despread,CPICH}\} = 2 \cdot a \cdot [\cos(q) + \sin(q)] + h_0 \tag{3.22}$$

Using (3.21) and (3.22), the channel estimation values *cos(q)* and *-sin(q)* can be written as

$$\mathrm{Re}\{\hat{R}_{despread,CPICH}\} + \mathrm{Im}\{\hat{R}_{despread,CPICH}\} = 4 \cdot a \cdot \cos(q) + h_0 \tag{3.23}$$

and

$$\mathrm{Re}\{\hat{R}_{despread,CPICH}\} - \mathrm{Im}\{\hat{R}_{despread,CPICH}\} = -4 \cdot a \cdot \sin(q) + h_0 \tag{3.24}$$

Note that (3.18) and (3.23) are different only in the constant gain term 2 and 4 and likewise (3.19) and (3.24). The second method based on the despread symbol doubles the coding gain compared with the first method based on the received signal before the dispreading operation. Note that the channel estimation values of *cos(q)* and *-sin(q)* have the same variable gain term a and AWGN h_0. The structure of the channel estimation based on method 2 is given in Figure 3.9.

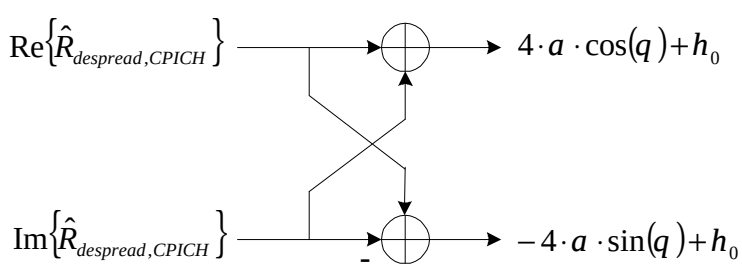


Figure 3.9: Channel estimation method 2 without a moving average

Figure 3.10 shows the simulation results on the performance of the channel compensation for the two different channel estimation methods without existence of the frequency offset. The two channel estimation methods are compared with two different mobile speeds, vehicular speed of 120 Km/h and the pedestrian speed of 5 Km/h. In both environments, the second method provides twice the coding gain over the first method. The performance improvement in the case of a low speed mobile is about 20 dB at a high SNR, which is significant. Since the Rayleigh fading due to the mobile speed is not

dominant, the gain factor of only 2 improves the performance tremendously as shown in Figure 3.10.

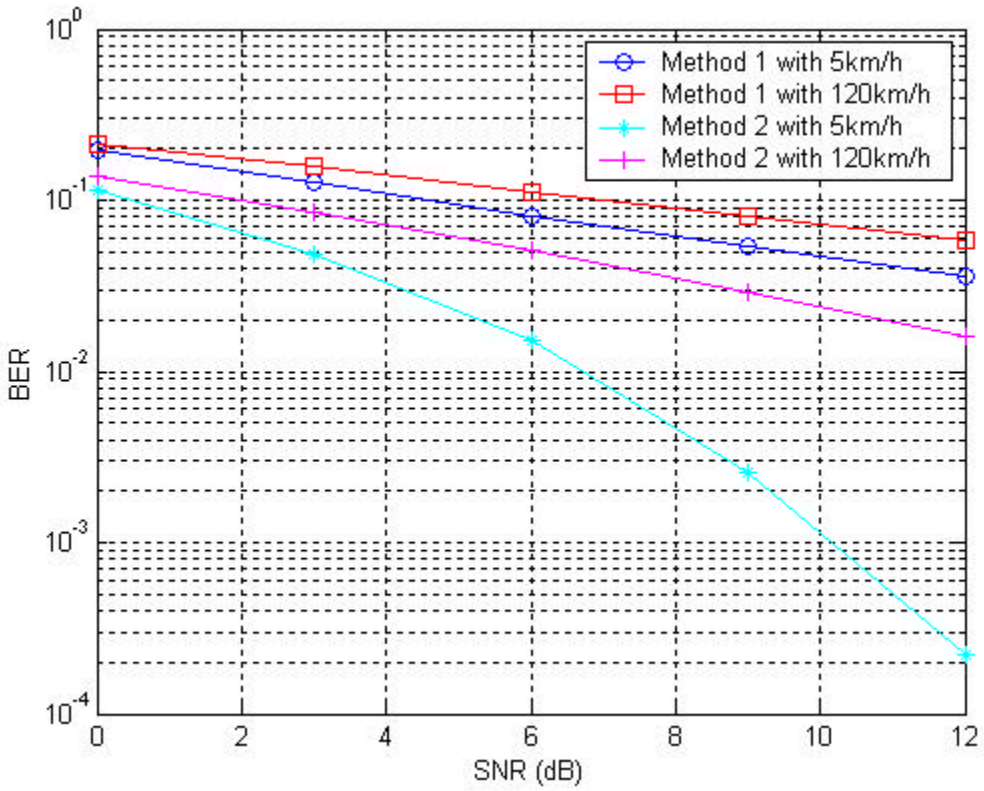


Figure 3.10 Channel estimation method comparison

Now, let us consider the elimination of the AWGN using the moving average scheme. The moving average scheme is shown in Figure 3.11. Ideally, a complete removal of the AWGN needs the moving average of an infinite number of symbols, which is impractical. Therefore, we need to select an appropriate number of symbols for the moving average that optimizes performance and hardware efficiency.

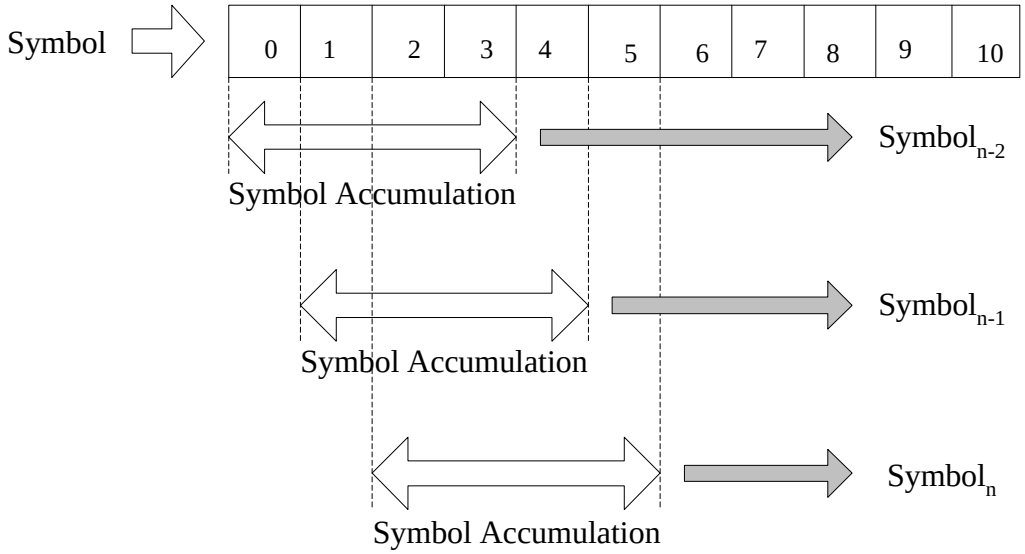


Figure 3.11: Moving average structure

As shown in Figure 3.12, two different positions are available for the moving average. The first position is after the dispreading operation and the second one is after the calculation of the channel estimation.

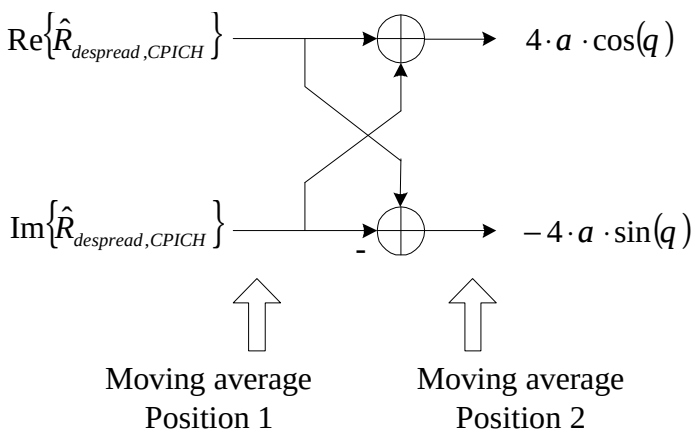


Figure 3.12: Two possible positions for the moving average in the channel estimation

In order to determine the number of symbols for the moving average operation, we simulated six different cases, in which the number of symbols for the moving average

is 2^n, $0 \le n \le 5$. The number of symbols is set to a power of 2 for low hardware complexity; in such a case a division operation is simply a shift right operation.

First, we estimated the performance of the moving average for Position 1 without the frequency offset. Figures 3.13 and 3.14 show the performances with six different moving average lengths. For both low speed and high-speed cases, the BER increases as the moving average length increases up to four pilot symbols. With moving average lengths larger than four, the performance degrades in the high-speed mobile case. This tendency is explained as the mobile speed increases the channel changes rapidly, but a long moving average length makes the channel estimator slow to respond to rapid channel change. Also, as the length of the moving average increases, the hardware complexity increases. Therefore, the moving average length of 4 pilot symbols seems a good compromise between the performances and hardware complexity for Position 1.

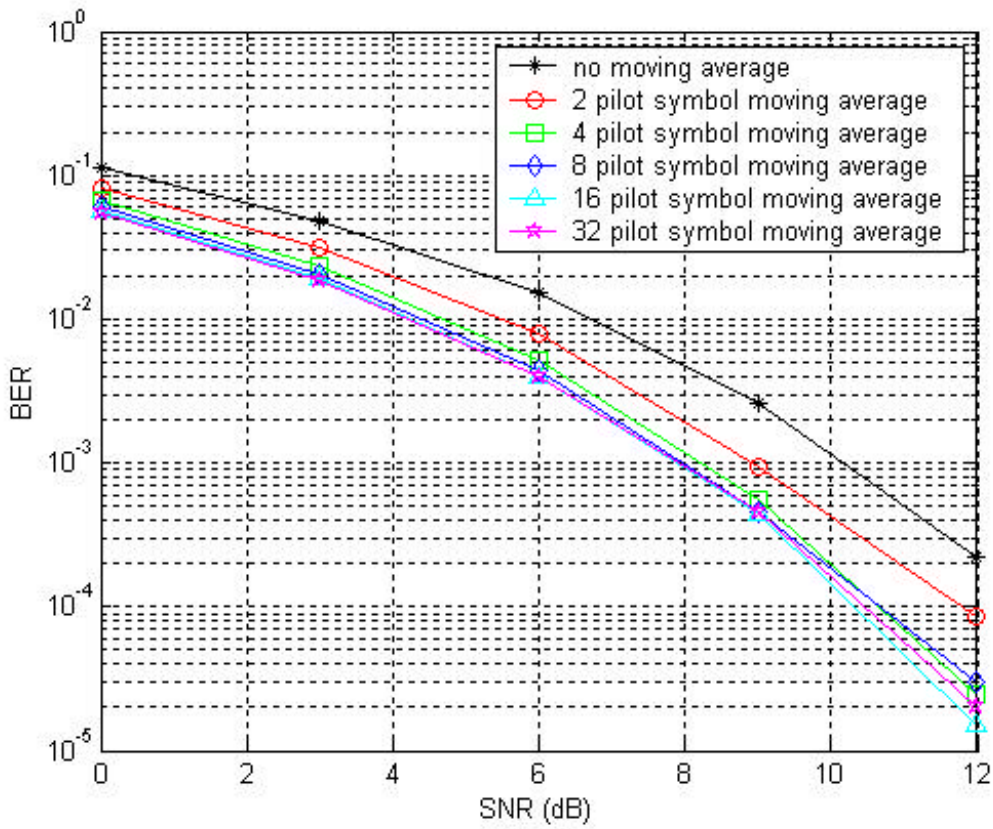


Figure 3.13: Moving average with the mobile speed of 5km/h in Position 1

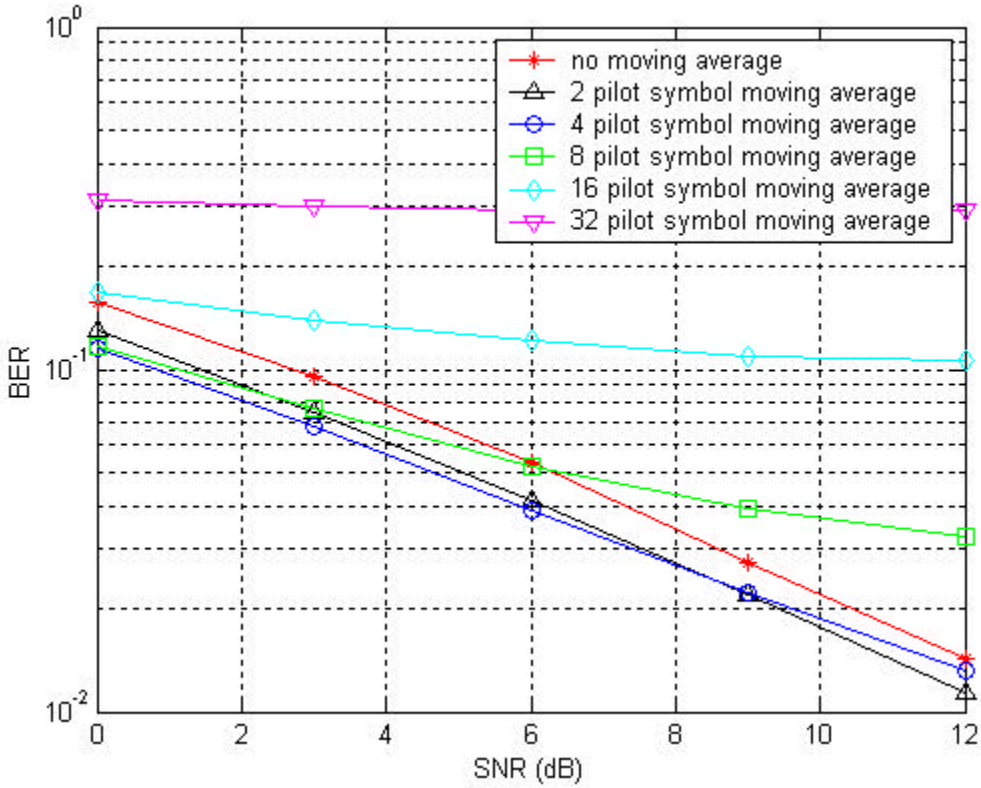


Figure 3.14: Moving average with the mobile speed of 120km/h in Position 1

Next, we estimated the performance for Position 2, and the simulation results are shown in Figures 3.15 and 3.16. As in the previous case, the moving average length of four seems a good choice for Position 2. We compared the performance of Position 1 and Position 2 for two mobile speeds under the moving average length four, and the result is given in Figure 3.17. As can be seen in Figure 3.17, Position 1 and Position 2 yield the same performance in terms of the BER.

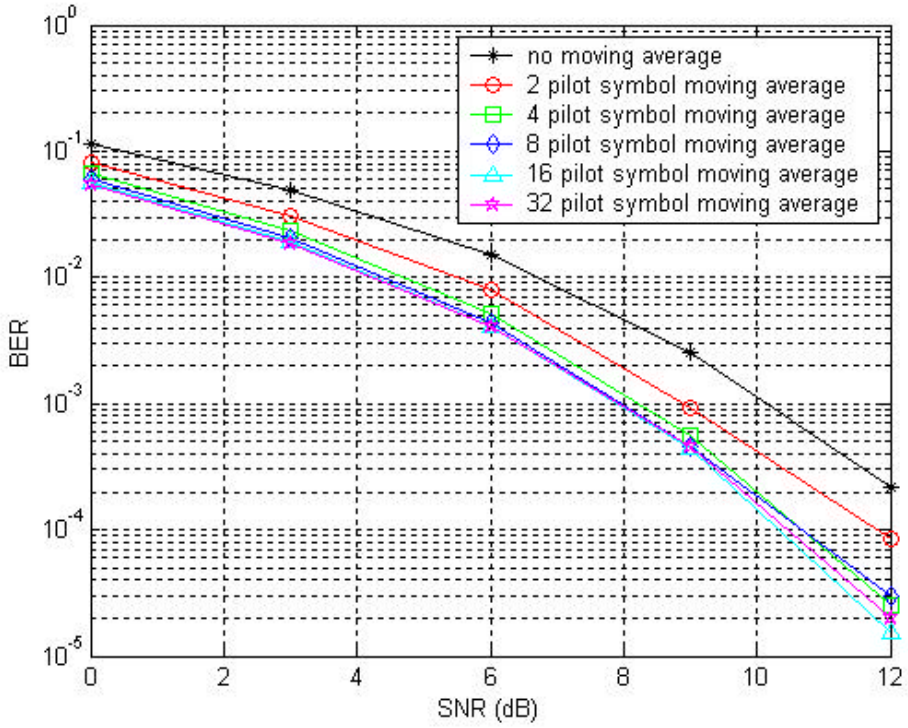


Figure 3.15: Moving average with the mobile speed of 5km/h at position 2

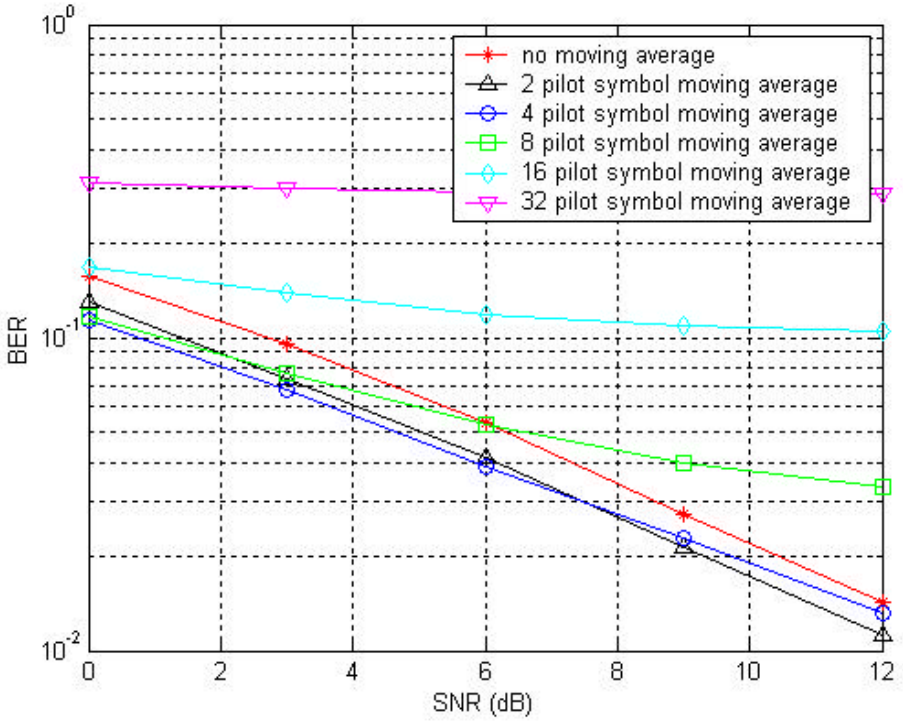


Figure 3.16: Moving average with the mobile speed of 120km/h at position 2

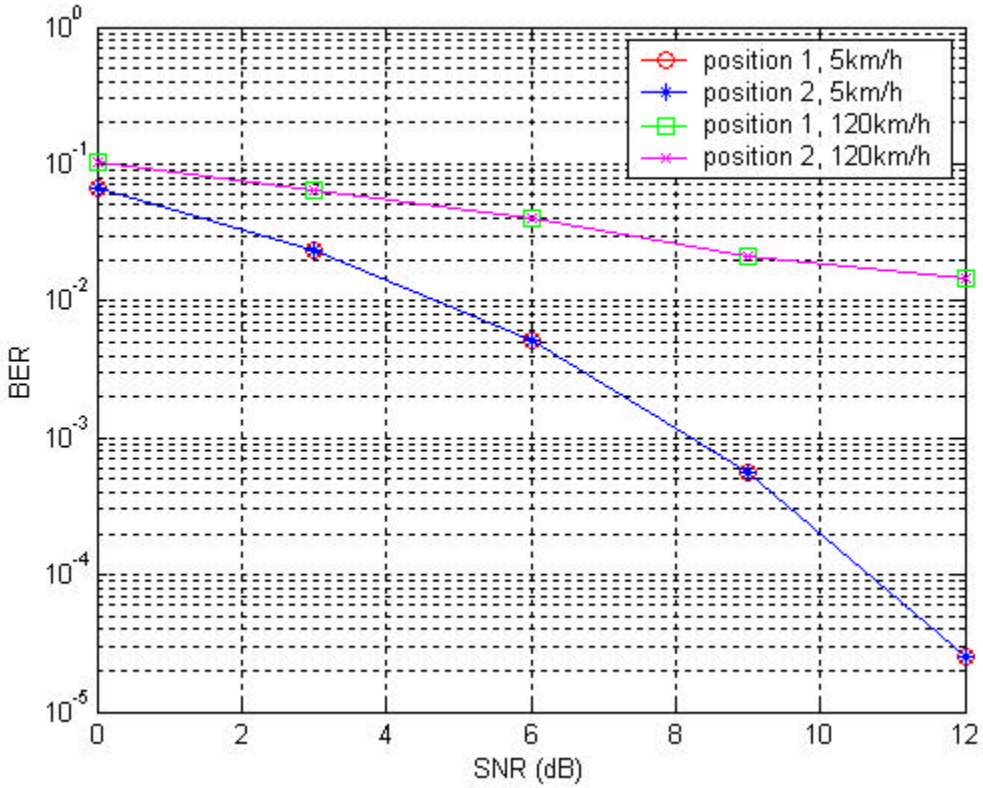


Figure 3.17: Performance comparison for the different moving average positions

Position 1 is chosen for our design due to simpler hardware as described next. First, the magnitude of the inputs to the moving average block is smaller for Position 1 that that of Position 2. This is because the inputs for the moving average for Position 2 are the sums of the inputs for the moving average block for Position 1. Second, the moving average block can be shared with the frequency offset estimator placed right after the dispreading block. Therefore, the moving average position of Position 1 reduces the circuit complexity. The resultant channel estimator that includes the moving average block is illustrated in Figure 3.18.

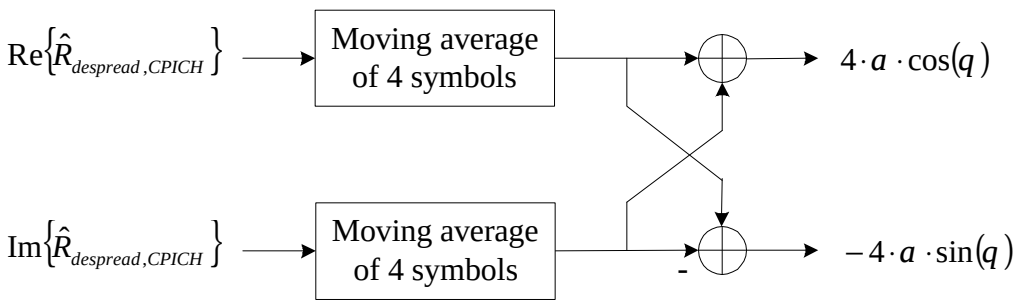


Figure 3.18: Channel estimator

3.2.5 Channel Compensation

Channel compensation operation is to multiply despread DPCH symbols with $[\cos(q) - j \cdot \sin(q)]$, which is obtained by channel estimation as in (3.23) and (3.24). Using (3.9), (3.23) and (3.24), the channel compensation result of the DPCH is

$$\begin{aligned} & R_{ch\_comp,DPCH} \\ & = R_{despread,DPCH} \times 4a \cdot [\cos(q) - j \cdot \sin(q)] \\ & = 2a \cdot (D_i + j \cdot D_q) \cdot \exp\{j(q+f)\} \times 4a \cdot \exp\{-j(q)\} + h_0 \\ & = 8a^2 \cdot (D_i + j \cdot D_q) \cdot \exp\{j(f)\} + h_0 \end{aligned} \tag{3.25}$$

As we can observe in (3.25), the channel compensation eliminates the effect of the channel phase rotation, i.e., the θ term. However, the frequency offset and AWGN still exist. The channel compensation process is simply a complex multiplication with an input $4a \cdot [\cos(q) - j \cdot \sin(q)]$ and is shown in Figure 3.19.

The channel compensation term $4a \cdot [\cos(q) - j \cdot \sin(q)]$ is obtained from the pilot channel, and this term is updated at the pilot symbol rate, i.e., every 256 chips. For example, if the spreading factor of the DPCH is four, the channel compensation term is updated at every 64 DPCH symbols, which corresponds to one pilot symbol.

The impact of the channel compensation under the Rayleigh fading channel model is shown in Figure 3.20. The improvement is greater for a mobile speed of 5 Km/h than one of 120 Km/h. Though the performance is improved through the channel compensation, it is still less effective than the performance in which only AWGN exists.

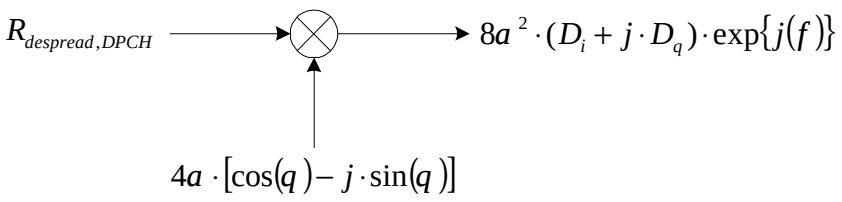


Figure 3.19: Channel compensator

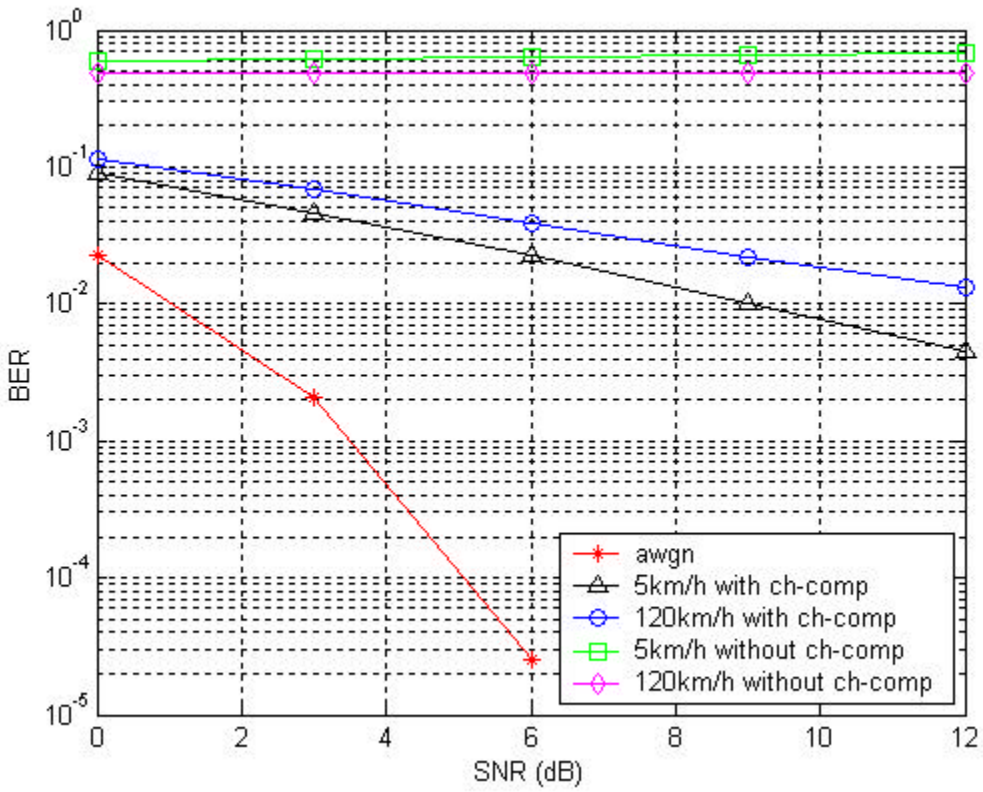


Figure 3.20: Performance of the channel compensation

3.2.6 Frequency Offset Estimation

Like the channel estimation, the frequency offset estimation also uses despread CPICH symbols. We assume that the moving average removes the AWGN completely. With (3.10), the despread CPICH symbols can be expressed as (3.26) after the moving average and by noting $P_i = P_q = 1$.

$$\hat{R}_{despread,CPICH}$$
$$= 2(1+j)\cdot\{a\exp(jq)\cdot\exp(jf)\}$$
$$= 2(1+j)\cdot[a\cdot\cos(q+f)+ja\cdot\sin(q+f)]$$
$$= 2a\cdot\{\cos(q+f)-\sin(q+f)\}+j\cdot 2a\cdot\{\cos(q+f)+\sin(q+f)\} \quad (3.26)$$

From (3.26), the frequency offset Df can be estimated using the current time n and the previous time n-1 pilot symbols as shown below.

$$\hat{R}_{despread,CPICH,n}\cdot\hat{R}^{*}_{despread,CPICH,n-1}$$
$$= \mathrm{Re}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Re}\{\hat{R}_{despread,CPICH,n-1}\}+\mathrm{Im}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Im}\{\hat{R}_{despread,CPICH,n-1}\}$$
$$+ j\cdot[\mathrm{Im}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Re}\{\hat{R}_{despread,CPICH,n-1}\}-\mathrm{Re}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Im}\{\hat{R}_{despread,CPICH,n-1}\}] \quad (3.27)$$

The real and imaginary terms are simplified as

$$\mathrm{Re}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Re}\{\hat{R}_{despread,CPICH,n-1}\}+\mathrm{Im}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Im}\{\hat{R}_{despread,CPICH,n-1}\}$$
$$= 4a^2\cdot\{\cos(q_n+f_n)-\sin(q_n+f_n)\}\cdot\{\cos(q_{n-1}+f_{n-1})-\sin(q_{n-1}+f_{n-1})\}$$
$$+4a^2\cdot\{\cos(q_n+f_n)+\sin(q_n+f_n)\}\cdot\{\cos(q_{n-1}+f_{n-1})+\sin(q_{n-1}+f_{n-1})\}$$
$$= 8a^2\cdot\cos(\Delta q+\Delta f) \quad (3.28)$$

$$\mathrm{Im}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Re}\{\hat{R}_{despread,CPICH,n-1}\}-\mathrm{Re}\{\hat{R}_{despread,CPICH,n}\}\cdot\mathrm{Im}\{\hat{R}_{despread,CPICH,n-1}\}$$
$$= -4a^2\cdot\{\cos(q_n+f_n)-\sin(q_n+f_n)\}\cdot\{\cos(q_{n-1}+f_{n-1})+\sin(q_{n-1}+f_{n-1})\}$$
$$+4a^2\cdot\{\cos(q_n+f_n)+\sin(q_n+f_n)\}\cdot\{\cos(q_{n-1}+f_{n-1})-\sin(q_{n-1}+f_{n-1})\}$$
$$= 8a^2\cdot\sin(\Delta q+\Delta f) \quad (3.29)$$

where $Dq = q_n - q_{n-1}$ and $Df = f_n - f_{n-1}$. Since the frequency offset Df has a fixed value and is very small, accumulation of (3.28) and (3.29) terms for several slots influences the accuracy of the frequency offset as shown in (3.30). Note that the gain term is eliminated by the division operation in the calculation of *tan(Df)*. Thus, the length of the accumulator can be as long as needed. In our design, one frame (150 pilot symbols) is assigned as the length of the accumulator.

$$\tan(\Delta f) = \frac{\sum 8a^2\cdot\sin(\Delta q+\Delta f)}{\sum 8a^2\cdot\cos(\Delta q+\Delta f)} = \frac{\sum\sin(\Delta q+\Delta f)}{\sum\cos(\Delta q+\Delta f)} \quad (3.30)$$

The accumulation process also removes the channel phase offset *Dq*, because the channel phase rotation is a random value. Thus, (3.30) can be rewritten as (3.31):

$$\tan(\Delta f) = \frac{\sum \sin(\Delta f)}{\sum \cos(\Delta f)} \tag{3.31}$$

From (3.31) the frequency offset value Df can be computed by the arc tangent operation as expressed in (3.32):

$$\Delta f = \tan^{-1}(\tan(\Delta f)) \tag{3.32}$$

Then, the angle frequency Dw is obtained as

$$\Delta w = \frac{\Delta f}{\Delta t} \tag{3.33}$$

where *Dt* is the symbol time interval of the DPCH. Finally, the frequency offset of each symbol can be computed as

$$f_n = \Delta w \cdot t_n \tag{3.34}$$

where $t_n = t_{n-1} + Dt$ and t_n represents the symbol time. Based on this result, the frequency offset estimation values of $cos(f_n)$ and $sin(f_n)$ can be obtained from a cosine/sine lookup table.

The structure of the frequency offset estimator is shown in Figure 3.21. Since the despread CPICH symbols have the AWGN, the moving average is needed. The moving average implemented in the channel estimator can be shared by the frequency offset estimator, and this is one reason the moving average block in the channel estimator is placed right behind the despreading. If the moving average position is select as the end of the calculation of channel estimation values, we may need a new moving average block for the frequency offset compensator, and this may cause an increase in circuit complexity. Also, the division operation and a lookup table in the frequency offset estimator is very complex in hardware.

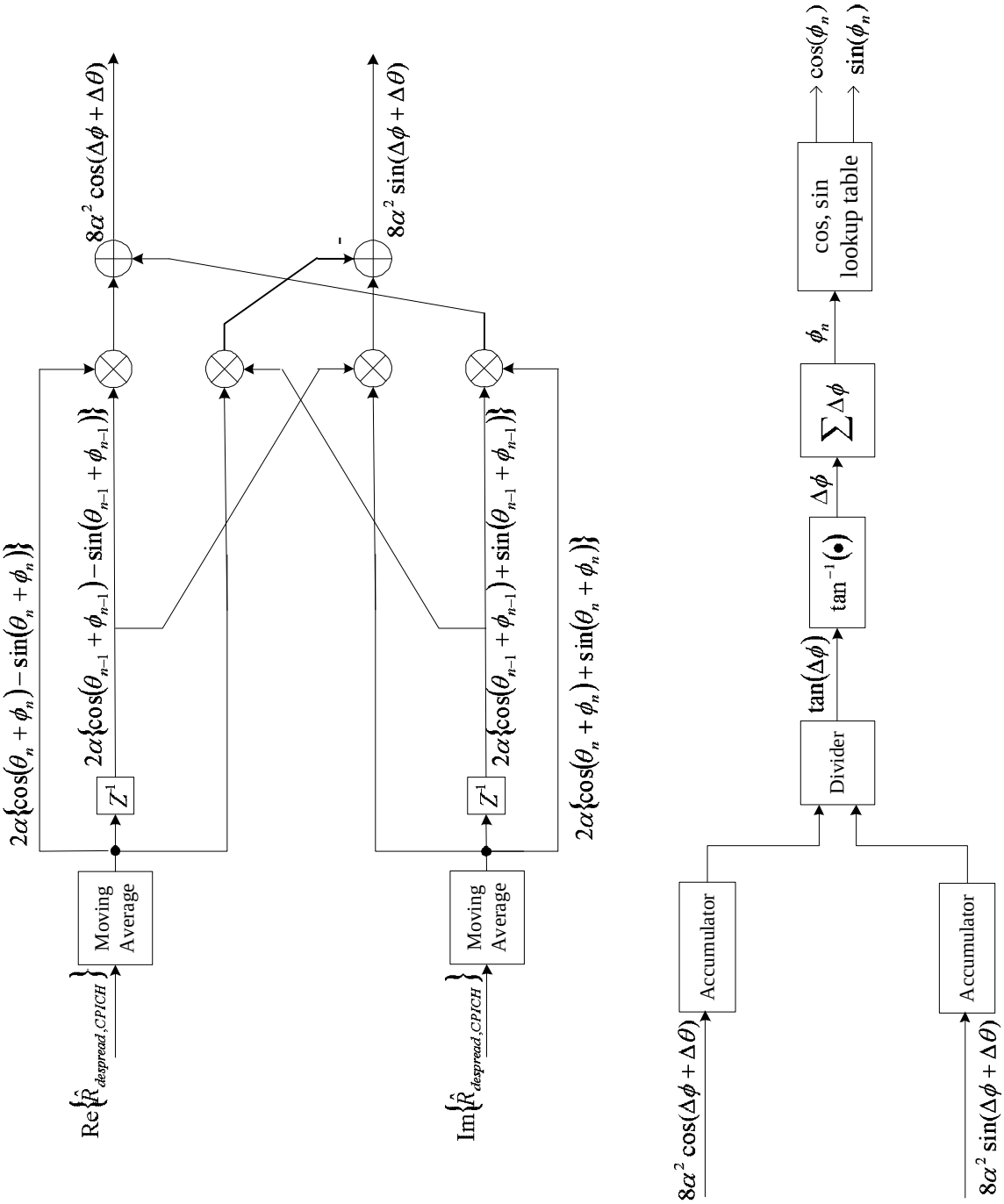


Figure 3.21: Frequency offset estimator

3.2.7 Frequency Offset Compensation

The operation of the frequency offset compensator is similar to the channel compensator. The structure is shown in Figure 3.22.

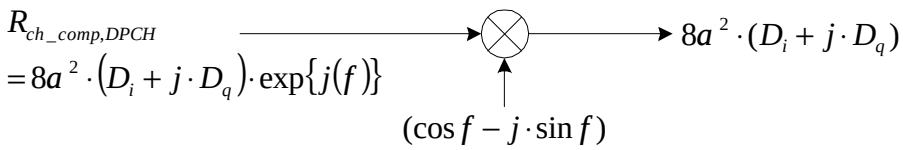


Figure 3.22: Frequency offset compensator

As shown in Figure 3.22, the frequency offset for the channel compensated DPCH is removed after the frequency offset compensation, and the desired DPCH symbols D_i and D_q are acquired with a gain term $8\alpha^2$. The gain term plays a role for the maximal ratio combiner.

3.2.8 Frequency Offset Compensation with a Channel Compensator

In the uplink, the pilot symbols are time-multiplexed with other control signals, such as TPC, FBI and TFCI. Thus, for an accurate estimation of the frequency offset, the channel phase rotation and the frequency offset cannot be processed at the same time. In the downlink, however, the pilot uses a separate channel, CPICH, so that it provides continuous control information. Therefore, this offers a possibility of compensating for the frequency offset and the channel phase rotation simultaneously. We propose a new method based on this possibility. We discuss our method below.

After the moving average, (3.10) is expressed as

$$
\begin{aligned}
& R_{desrpead,CPICH,avg} \\
& = 2(P_i + jP_q) \cdot \{a \exp(jq) \cdot \exp(jf)\} \quad \text{(Note } P_i = P_q = 1\text{)} \\
& = 2(1 + j) \cdot [a \cdot \cos(q + f) + ja \cdot \sin(q + f)] \\
& = 2 \cdot a \cdot [\{\cos(q + f) - \sin(q + f)\} + j\{\cos(q + f) + \sin(q + f)\}]
\end{aligned} \tag{3.35}
$$

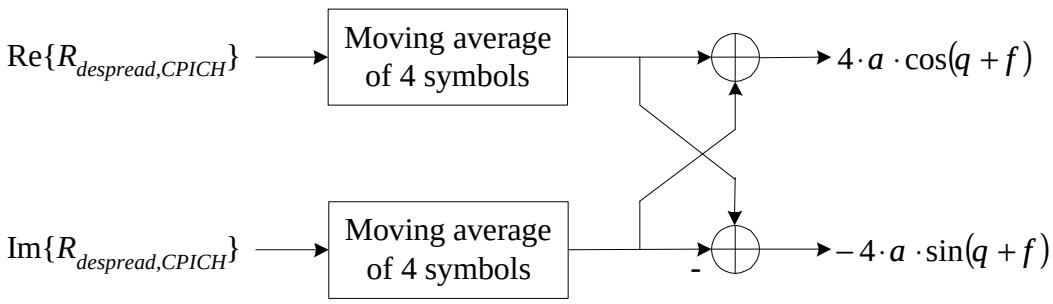
From (3.35), *cos(q+f)* and *-sin(q+f)* terms are obtained as

$$\mathrm{Re}\{R_{despread,CPICH,avg}\} + \mathrm{Im}\{R_{despread,CPICH,avg}\} = 4 \cdot a \cdot \cos(q+f) \tag{3.36}$$

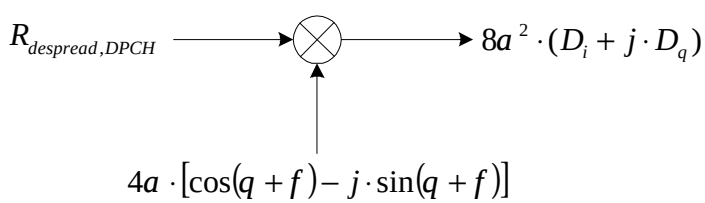
and

$$\mathrm{Re}\{R_{despread,CPICH,avg}\} - \mathrm{Im}\{R_{despread,CPICH,avg}\} = -4 \cdot a \cdot \sin(q+f) \tag{3.37}$$

Based on (3.36) and (3.37), we can compensate the channel phase rotation and the frequency offset simultaneously as shown in Figure 3.23.



(a) Channel / frequency offset estimator



(b) Channel / frequency offset compensator

Figure 3.23: Simultaneous Channel / frequency offset compensator

The frequency offset compensation performance by the combined compensator is examined for a mobile speed of 120 Km/h. Figure 3.24 shows the performance of the combined compensator for the two frequency offsets of 200 Hz and 400 Hz. The BER's for both frequency offsets are close to that of the zero frequency offset in the figure. The combined compensator saves the hardware substantially without degrading the performance in the downlink.

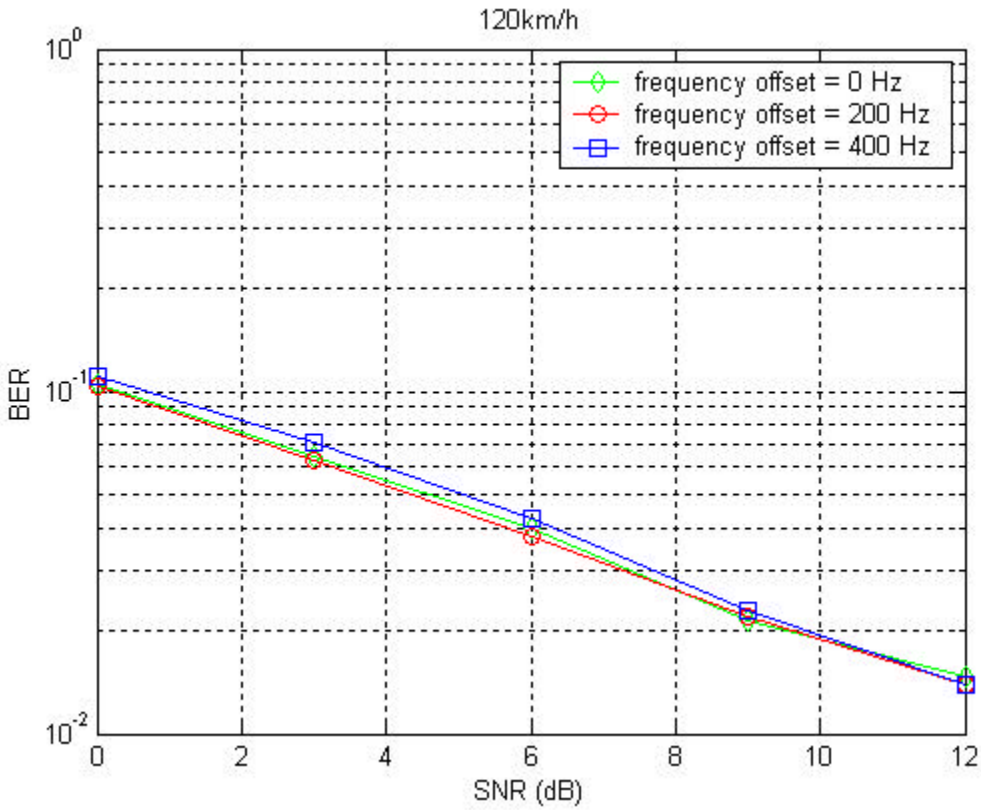


Figure 3.24: Frequency offset compensation by the combined compensator

3.2.9 Power Estimation

The combiner of a rake receiver adds the multipath signals processed by individual fingers provided the power level of a multipath signal exceeds a threshold value. A weak multipath signal is dropped by the combiner to avoid the adverse effects of weak signals to the overall performance. The process requires the estimation of signal power for each multipath.

Power estimation for a multipath is performed with the channel / frequency offset compensated CPICH symbols. (3.38) expresses the power estimation equation.

$$P_D = \sqrt{\left(\mathrm{Re}\{R_{comp,CPICH}\}\right)^2 + \left(\mathrm{Im}\{R_{comp,CPICH}\}\right)^2} = \sqrt{\left(8a^2 \cdot D_i\right)^2 + \left(8a^2 \cdot D_q\right)^2} \qquad (3.38)$$

Figure 3.25 shows the process.

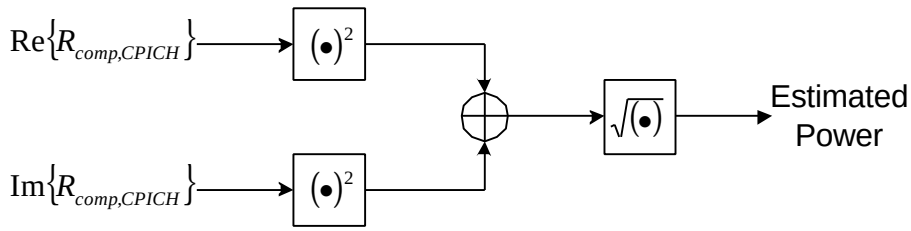


Figure 3.25: Power estimator

Though the actual estimated power should be $\left(\mathrm{Re}\{R_{comp,CPICH}\}\right)^2 + \left(\mathrm{Im}\{R_{comp,CPICH}\}\right)^2$, the square root operation reduces the circuit complexity in two areas. First, the equation given in (3.38) can be approximated to an equation (3.39) that does not need multiplications.

$$\hat{P}_D = Max\left[\mathrm{Re}\{R_{comp,CPICH}\}, \mathrm{Im}\{R_{comp,CPICH}\}\right] + \frac{1}{2} Min\left[\mathrm{Re}\{R_{comp,CPICH}\}, \mathrm{Im}\{R_{comp,CPICH}\}\right] \tag{3.39}$$

As shown in Figure 3.26, this approximate value is close to the accurate value. When the compensation is perfect, which means that the AWGN is perfectly removed and the phase rotation and the frequency offset are completely compensated, the approximate value has 6 % of error, on average. In the power estimation, the relative accuracy, rather than the absolute accuracy, is important. Hence, the approximation is practical.

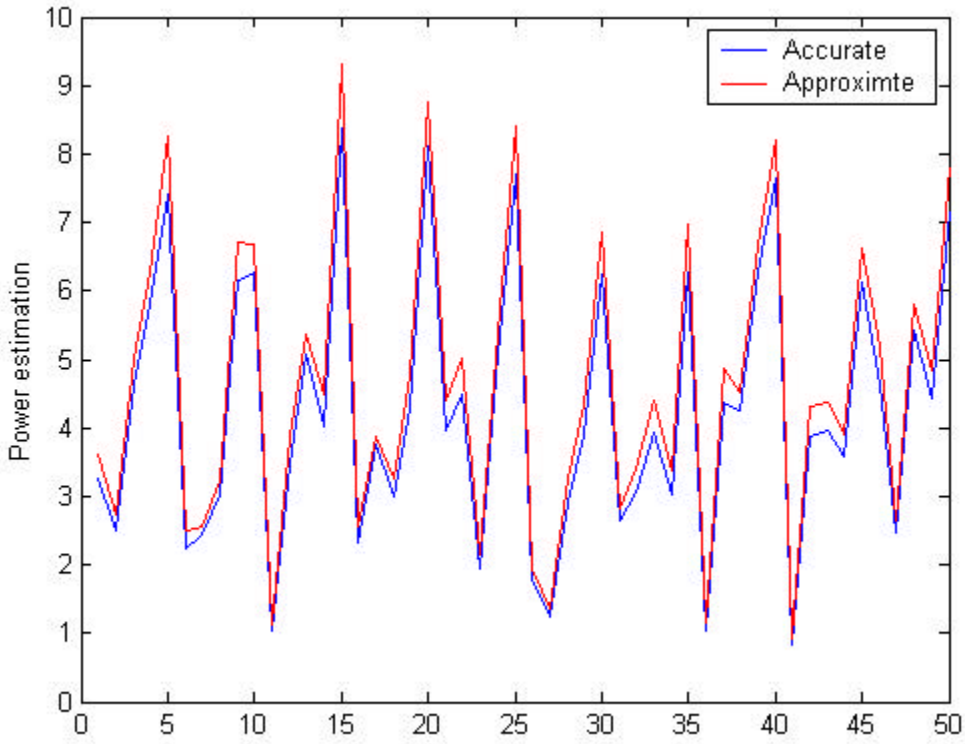


Figure 3.26: Comparison on the power estimation methods

Second, the approximate value reduces the number of bits for the representation of the estimated power. Suppose that the square root is not used. The number of bits required to represent the value is $n+2$ for an estimated power of 2^n. However, by taking the square root value of 2^n, the estimated power can be expressed with only $\frac{n}{2}+2$ bits, which is almost half for a large n. We use the approximated power estimation method given in (3.39) for the hardware implementation.

3.2.10 Time Tracker

As described in Chapter 2, the transmitted signals are pulse shaped with a root-raised cosine filter. The received signal after the pulse shaping is over-sampled eight times. A time tracker is responsible for tracking the exact sampling point by advancing

or delaying the sampling points. A rake receiver receives the timing information from a cell searcher at the ½-chip interval. Then, the time tracker performs fine-tuning to compensate for the Doppler effect with the interval of 1/8-chip using the pilot symbols of the CPICH.

A time tracker updates the tracking information for every frame in the increment / decrement of a 1/8-chip interval. Let us check whether the number of the samples per chip (which is eight) and the updating interval (which is one frame) are sufficient to track the timing variation due to the Doppler effect. Suppose that the maximum mobile speed is 250 Km/h, which is 69.44 m/sec. It takes 231.5 nsec for the signal to travel 69.44 meters, which corresponds to 0.89 chip as shown in the following.

$$\frac{69.44m}{3 \cdot 10^8 m/\sec} = 231.5n\sec \tag{3.40}$$

$$\left(3.84 \cdot 10^6 chips/\sec\right) \times \left(231.5 \cdot 10^{-9} \sec\right) = 0.89\, chip \tag{3.41}$$

Also, the maximum signal change happens when the mobile travels toward the base station. As a result, the signal changes only 1.78 chips/sec, that is 0.0178 chips/frame. Therefore, the maximum timing update of 0.125 chips/frame is sufficient to keep track of the timing variation caused by the Doppler effect.

Figure 3.28 shows a pulse shaped received signal for one chip duration, which is about 260.4 nsec. The eight points on the waveform indicate the sampling points. Three consecutive samples among the eight samples are called early, on-time, and late, and are used to calculate the powers of the late and early pilot symbols. The objective is to place the on-time sample at the peak of the chip waveform through advancement / delay of the sampling points. If the power of the early symbol is greater than the power of the late symbol, as shown in Figure 3.28 (a), the sampling timing is delayed by 1/8-chip. On the other hand, if the late symbol has a larger power level than the early symbol, as shown in Figure 3.28 (b), the timing should be advanced by 1/8-chip.

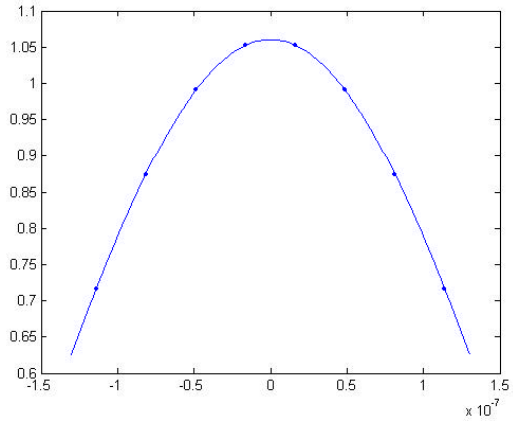
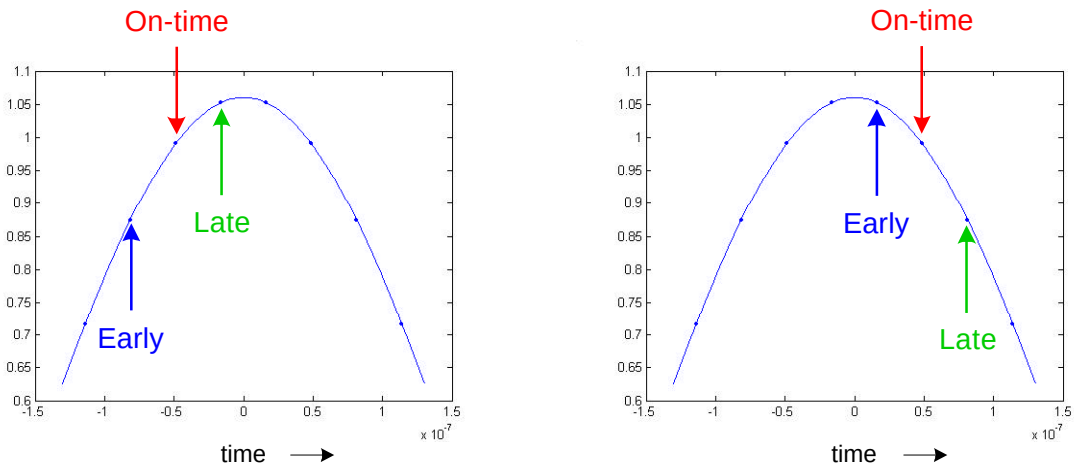


Figure 3.27: Pulse Shaped Chip



(a) Timing should be slowed

(b) Timing should be sped up

Figure 3.28: Early and late timing comparison

Figure 3.29 shows the structure of the time tracker. The time tracker receives the early and late despread CPICH pilot symbols, and calculates the power of each symbol. The difference of the early and late powers is applied to a second–order loop filter. The loop filter behaves as a lowpass filter, which removes any burst noise. Then the accumulator sums up the difference during one frame, and the accumulated difference is compared with a threshold value. If the accumulator output is positive and exceeds the threshold, the sampling timing is advanced by a 1/8-chip interval. If the accumulator

output is negative and its absolute value exceeds the threshold, the sampling timing is delayed by a 1/8-chip interval. The result is sent to the control block to generate the necessary control signals, which are applied to the clock generator. Details about the clock generator are covered in Chapter 4.

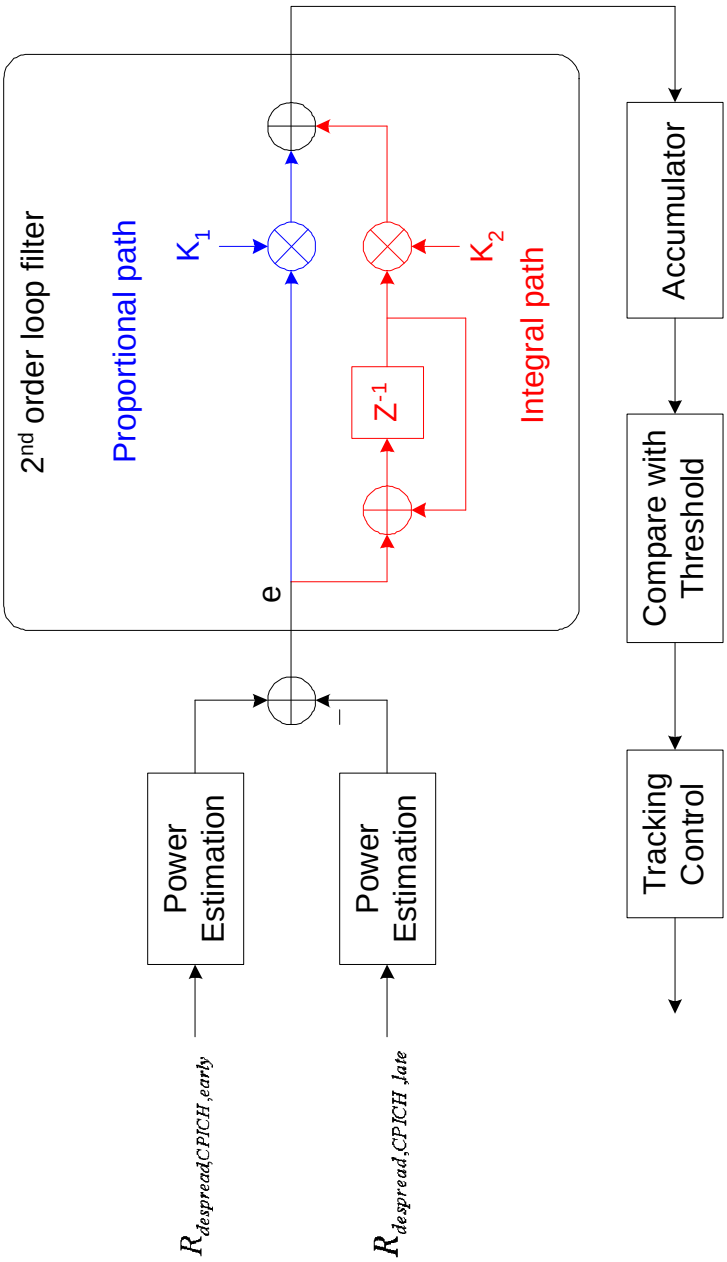


Figure 3.29: Time tracker

The second-order loop filter in the time tracker consists of two paths, which are a proportional path and an integral path. From control theory, it is known that a proportional path tracks out a phase error, but not a frequency error. A loop filter containing an integral path is needed to track out a sampling frequency error [Lit01]. The loop filter in Figure 3.29 has a proportional path and an integral path for these purposes.

To determine the two coefficients, K_1 and K_2, we use an equivalent linear model of the loop filter that is shown in Figure 3.30. The timing error e, which is proportional to the difference of the power between the early and late symbols, is represented as

$$e = e_0 \frac{\left(1 - z^{-1}\right)^2}{1 + \left(K_1 - 2\right)z^{-1} + \left(1 - K_1 + K_2\right)z^{-2}} \tag{3.42}$$

Coefficients K_1 and K_2 determine the loop bandwidth. If the loop bandwidth is high, the time tracker tracks the sample drift rapidly, but the time tracker can cause an unnecessary increment or decrement in timing. If the loop bandwidth is low, the tracking speed is slow, however, it eliminates the previous problem. The calculated loop bandwidths for various coefficient sets are given in Table 3.1.

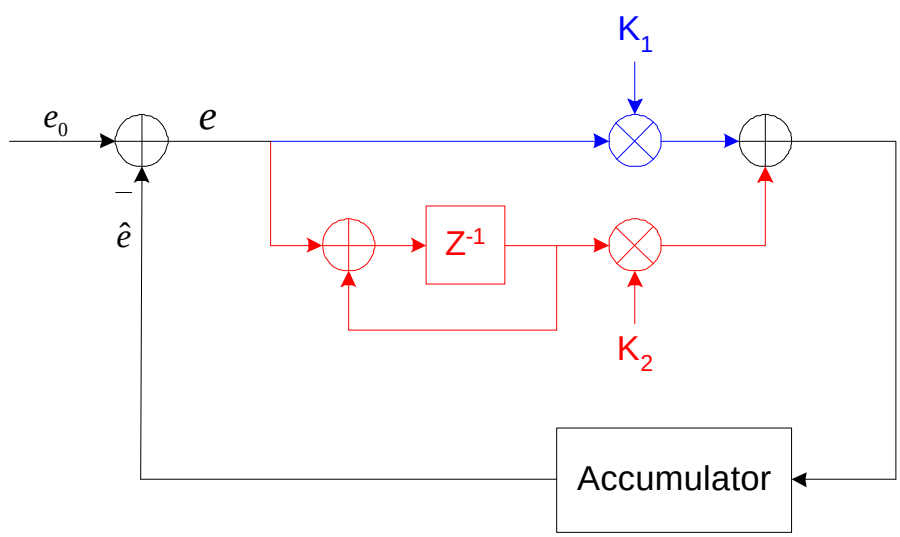


Figure 3.30: Equivalent linear model of the loop filter

Table 3.1: Loop bandwidth accroding to K_1 and K_2 [Neo00]

Loop bandwidth	K_1	K_2
13	2^{-3}	2^{-13}
30	2^{-2}	2^{-13}
60	2^{-1}	2^{-13}
120	2^{0}	2^{-13}

Since the loop filter behaves as a low pass filter, a straightforward low pass filter may be used instead of the loop filter. In this case, the maximum input frequency for the low pass filter should be 150 KHz, because the power difference can change its sign at most 150 (=38400/256) times during one frame period. Therefore, if the cutoff frequency of the low pass filter is lower than 150 KHz, the time tracker removes unwanted noise.

3.2.11 Deskewer

A deskewer is a series of buffers that compensate for multipath delays. The deskewer is not explicitly implemented in our simulation. Details on the deskewer are discussed in Chapter 4.

3.2.12 Combiner

A combiner adds up the multipath signals of the fingers. As noted earlier, the combiner discards weak finger outputs. Since we employ maximal ratio combining, the energy level of a finger output is weighted during combining. Therefore, a stronger signal has more impact on the decision process.

3.2.13 Finger structure

Figure 3.31 shows the overall structure of a finger whose components are covered in previous sections. Early and late CPICH signals are despread for time tracking, and on-time CPICH signals are despread for channel estimation and power estimation. The

DPCH signal is despread, and then compensated to eliminate the effect of the Rayleigh fading and the frequency offset. The resulting DPCH symbols are sent to the deskewer to compensate the multipath delays for combining. The result of power estimation is sent to the combiner to determine whether the deskewed data symbol is combined or not.

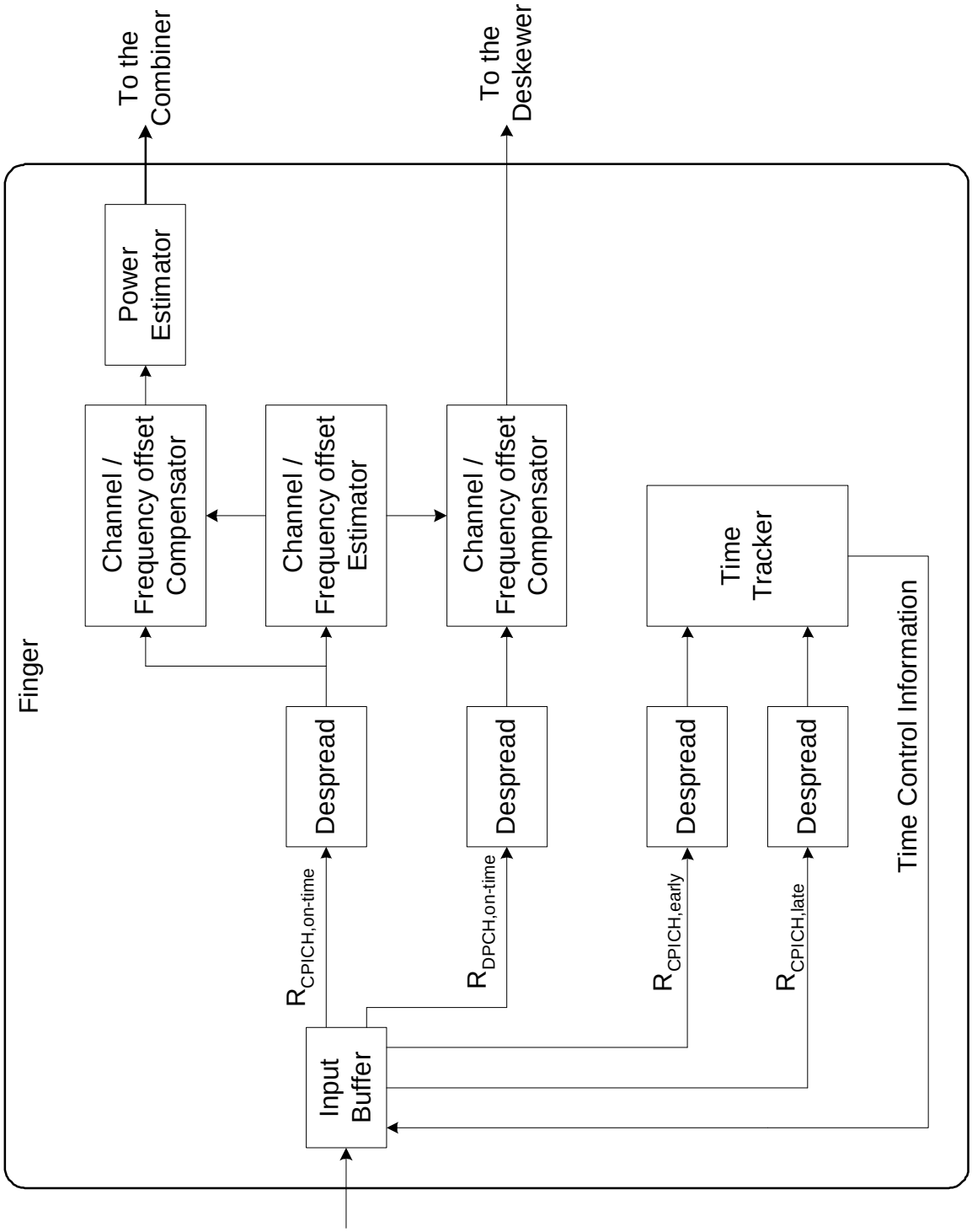


Figure 3.31: Structure of our finger

Chapter 4 VHDL models

Based on the specifications of WCDMA systems and the MATLAB simulation results described in Chapter 3, we implemented a rake receiver in VHDL. The VHDL models for the rake receiver include:

- demodulators,
- a channel-frequency (CF) estimator,
- a compensator,
- a frequency offset estimator,
- a power estimator,
- a time tracker,
- a deskewer,
- a combiner,
- a SRAM controller,
- a clock generator,
- a control signal generator,
- OVSF code generators,
- scrambling code generators,
- an interface unit

In the VHDL models, the deskewer, the combiner, and the SRAM controller are combined together into a deskew-combiner. The OVSF code generator, the scrambling code generator, and the interface unit are not discussed, as they are not the focus of this thesis.

Figure 4.1, which is similar to Figure 3.5 in section 3.2, shows a rake receiver structure, which does not include the interface unit and external memory. The rake receiver has four fingers to process four multipaths. The received train of chips is stored in the input buffer, and three consecutive chips, which are early, on-time and late chips, are applied to the fingers simultaneously. Each finger processes a multipath, and the deskew-combiner generates the combined symbol sequence from the finger outputs. In the following subsections, the operation of internal blocks is explained.

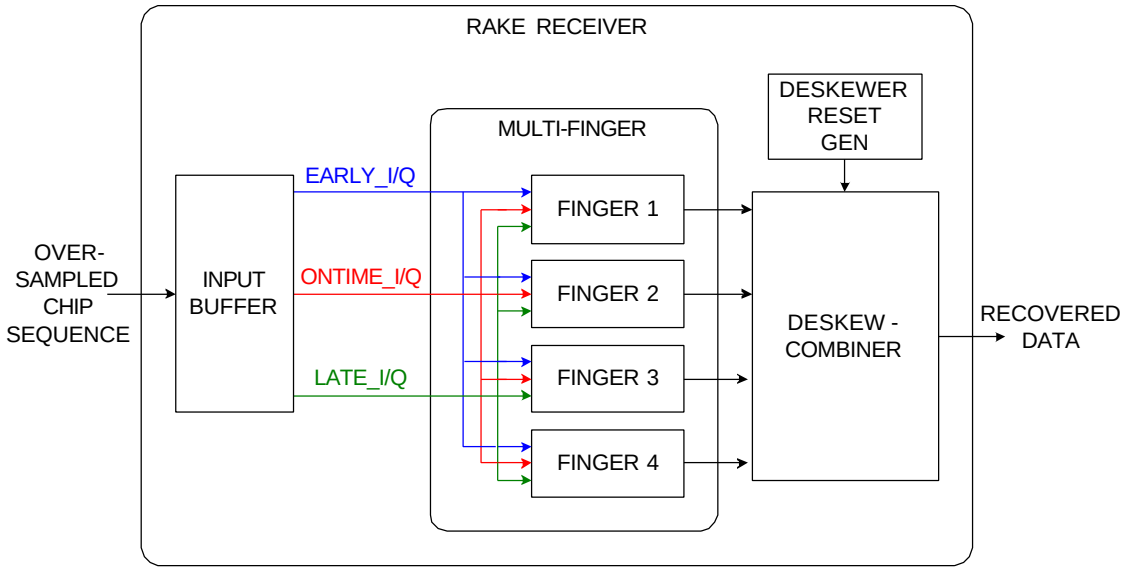


Figure 4.1: Rake receiver structure

4.1 Input buffer

The input buffer has two groups of shift registers as shown in Figure 4.2, one of which is for the I channel and the other for the Q channel. The rake receiver employs a 6-bit ADC to quantize the received signals, which is typical for WCDMA systems. The ADC of the receiver samples the received signal at a sampling rate of eight times the chipping rate, the input buffer receives the input sequence at every 1/8 chip and stores the data in the late register first, then shifts the existing data to the on-time and early registers.

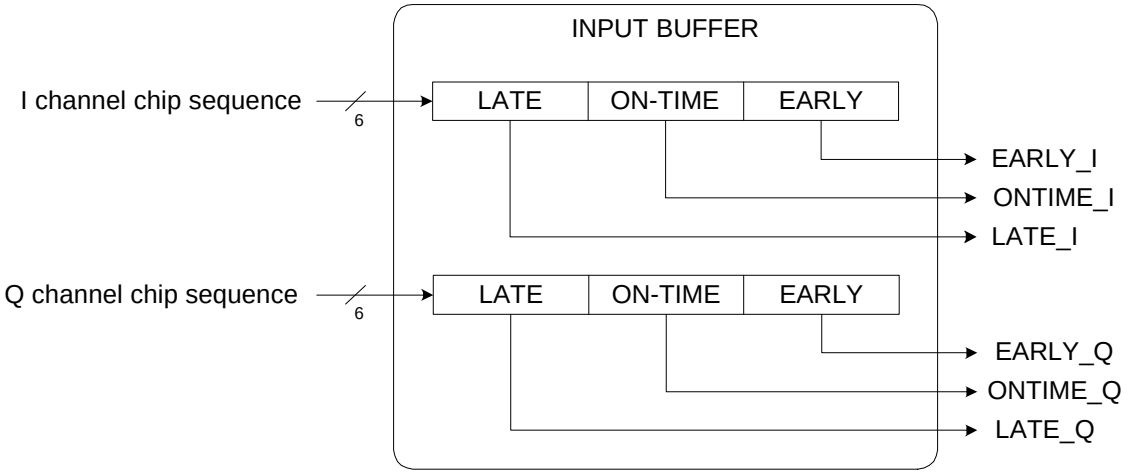


Figure 4.2: Input buffer

4.2 Finger

Figure 4.3 shows the structure of a finger. Each finger receives the early, on-time and late samples. The I/O signals, including clock signals and control signals for each component, are discussed in the following.

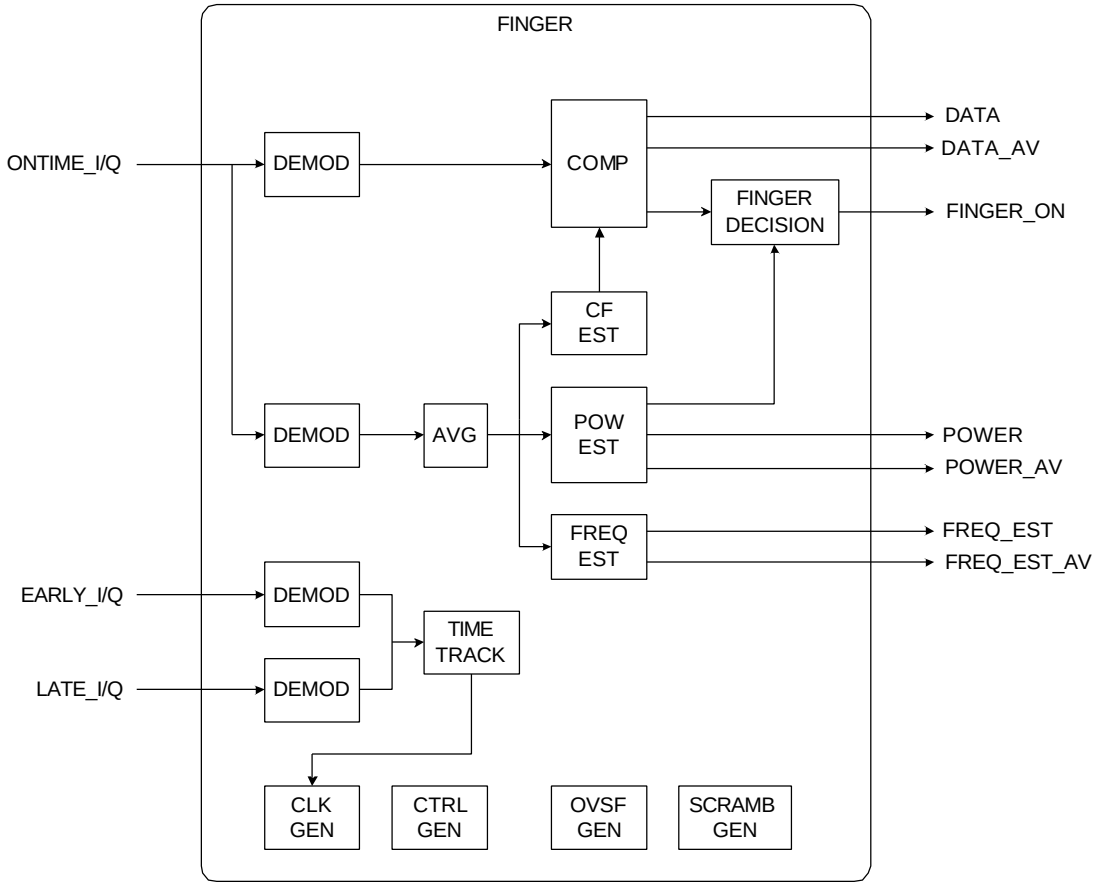


Figure 4.3: Internal structure of a finger

4.2.1 Clock generator (CLK_GEN)

The clock generator generates a reset signal for internal blocks, the clock signal at the chipping rate and four control signals for the blocks of the finger. Four factors determine the time offset value for the clock. They are a 10 msec synchronization signal, the time offset determined by the base station, the multipath delay, and the fine time tracking signal. A mobile receives a synchronization signal from the base station through the synchronization channel in every frame, i.e., at every 10 msec, and it is named SYNC_10M in our VHDL code. The time-offset value is the difference between the base station and the mobile covered by the base station, and it is in a 1/8-chip timing resolution. Also, the cell searcher computes the multipath delay in ½-chip timing accuracy. The clock generator receives the combined time offset, TIMEOFF_SC, with

the resolution of 1/8-chip from the cell searcher, and the combined offset is the sum of the time offset of the base station and path delay. The time tracker sends the tracking result LEAD_LAG to the clock generator. The starting point of the frame is calculated from these three signals, SYNC_10M, TIMEOFF_SC, and LEAD_LAG.

Since the finger starts the demodulation at the calculated frame starting point, the clock generator generates main clock signals in advance of the start of a frame for initialization and preparation for the demodulation. The detailed operations of the clock generator are described in the following sections.

4.2.1.1 I/O signals of the clock generator

Figure 4.4 shows the I/O signals of the clock generator, and Table 4.1 gives the role of each I/O signal. The clock generator receives eight inputs and sends six output signals. RESET is the reset signal for the entire rake receiver, and CHIPX8 is the clock signal from the local oscillator with a frequency of 30.72 MHz. One CHIPX8 cycle corresponds to the 1/8 of chip duration. SYNC_10M is the synchronization signal that ticks at every 10 msec. START_USER and FINGER_LOCK determine whether the mobile is turned on whether and the particular finger is locked for processing, respectively. Therefore, both signals should have the value ‘1’ to operate a finger, which signals the internal signal START to ‘1’. TIMEOFF_FAIL_RECOVER is a user’s optional setting for the recovery of the failed time offset; and the failure of the time offset is discussed in detail later. As described earlier, TIMEOFF_SC is the combination of the multipath delay of the cell searcher and the time-offset between a base station and the mobile, and LEAD_LAG is the time tracking result.

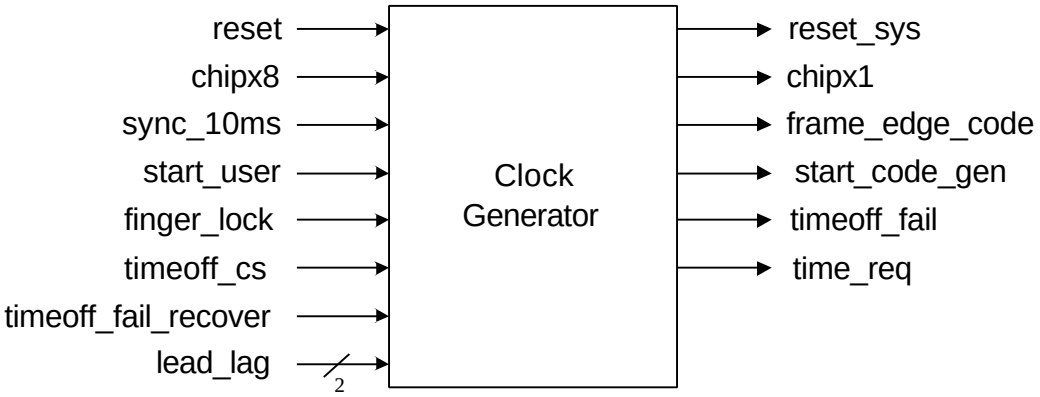


Figure 4.4: Clock generator

Table 4.1: I/O signals in the clock generator

Signal name	**I/O**	**Bit**	**Function**
Reset	I	1	Resets the entire system
Chipx8	I	1	Main clock from the local oscillator (30.72 MHz) which is eight times faster than the chipping rate
Sync_10m	I	1	10 msec synchronization signal from the synchronize channel
Start_user	I	1	Active high, when there is a call request and the rake receiver in the mobile is turned on to receive the signal
Finger_lock	I	1	States if the finger is assigned to a multipath signal or not
Timeoff_fail_recover	I	1	User setting for the recovery option of the time offset failure
Timeoff_sc	I	1	Time offset from the cell searcher. This includes the time-offset assigned by a connected base station
Lead_lag	I	2	Fine time tracking resulting form the time tracker
Reset_sys	O	1	Reset signal for the blocks of the finger
Chipx1	O	1	Chip clock (3.84 MHz)
Frame_edge_code	O	1	Frame edge for the code generators
Start_code_gen	O	1	Enable signal for the code generator
Time_req	O	1	Requests the fine time tracking result at the end of every frame
Timeoff_fail	O	1	Interrupt signal to DSP as the result of a failure to resolve the time-offset

4.2.1.2 Finite state machines

The operation of the clock generator is controlled with a finite state machine consisting of five sub state machines. We start with the main state machine, which is labeled as *operation state machine*.

4.2.1.2.1 Operation state machine

Figure 4.5 shows the *operation state machine*. To start the operation of the finger upon the arrival of the first chip of a frame, we need to resolve the starting point of the first CIHIPX1 in advance based on the received timing information such as SYNC_10M, TIMEOFF_SC and LEAD_LAG. The first CHIPX1 clock should be generated eight CHIPX8 clock cycles before starting demodulation, to initialize the system and generate the OVSF codes and scrambling codes which are used for the first data chip. Since the calculation of the time offset value typically takes seven CHIPX8 clock cycles, the calculation process should start 14 CHIPX8 cycles before the frame edge to reflect the proper time offset delay from the frame edge.

The frame counter, F_CNT, counts the number of 1/8-chips during one frame from the detection of the synchronization signal. F_CNT counts from zero to 307199 (38400*8-1). Therefore, the *operation state machine* makes its state transition, from the initial state INITIAL to TIMEOFF_REQ state, when F_CNT has the value of 307185 (307199-7-7). This is 14 CHIPX8 cycles ahead of the next expected SYNC_10M signal detection. The following state changes are shown in Figure 4.5. In the LOAD_TIMEOFF state, the counter for the time-offset, TIMEOFF_CNT, is set to zero, and the user setting for failure to recover, TIMEOFF_FAIL_RECOVER, and time tracking result, LEAD_LAG, are loaded. After loading TIMEOFF_SC, if a cell searcher time-offset is different from the previous one, the flag for the use of time tracking result, TRACK_OFF, is set to ‘1’. The desired time offset value is calculated in the CALC_TIMEOFF state. The detailed time offset calibration method is covered in Section 4.2.1.3. In the COUNT_TIMEOFF state, the value of TIMEOFF_CNT increases by one at every CHIPX8 until its value is equal to the calculated time offset value. When the time-offset counter reaches the same value with the calculated time-offset value, the

state changes to CLK_START to launch CHIPX1 generation. Specifically, the first generated CHIPX1 clock is for the initialization and code generation of the code generators. Then, the state is set back to INITIAL. All of the described operations, including the state transitions, are effective only when the START signal is high.

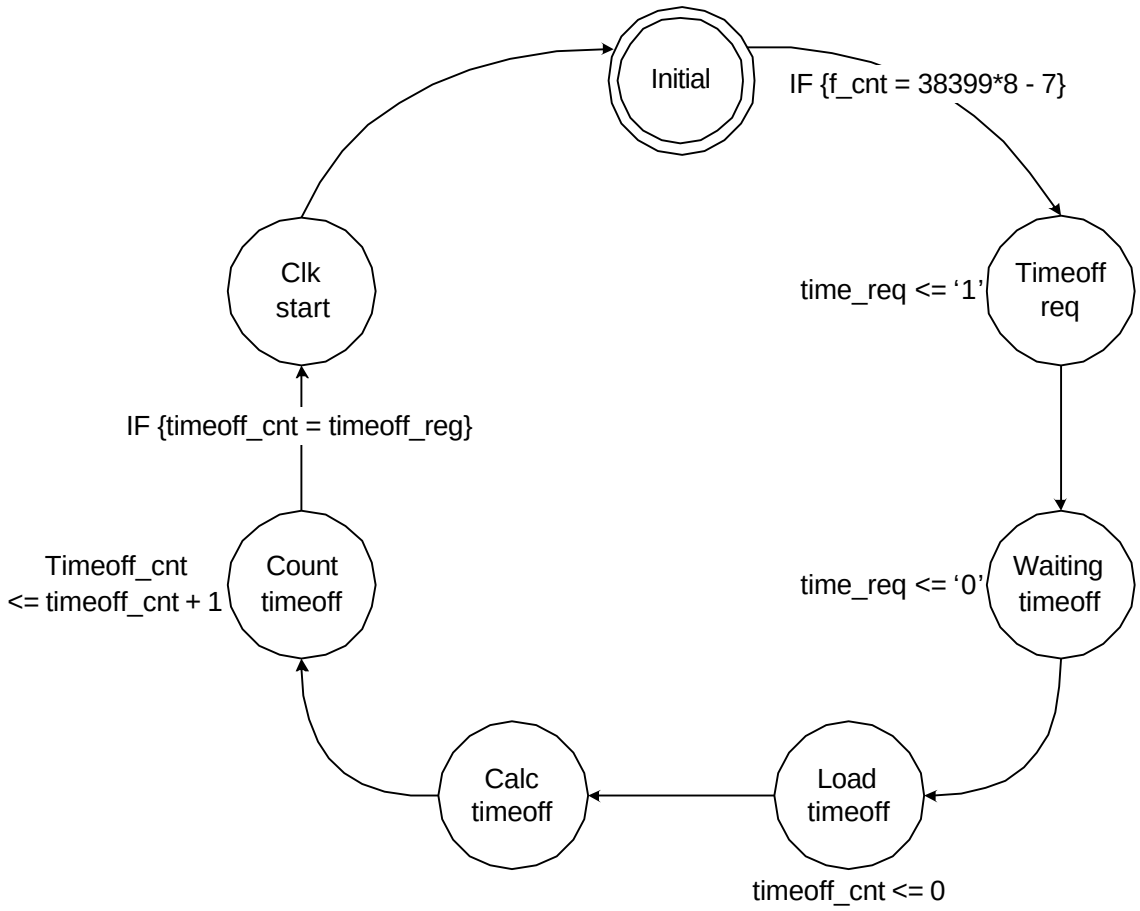


Figure 4.5: Operation state machine

4.2.1.2.2 Frame state machine

The finger receives a synchronization signal as a reference for the timing. The *frame state machine* controls the frame counter for the counting of CHIPX8 cycles according to the detection of the SYNC_10M signal. Figure 4.6 shows the *frame state machine*.

In the INITIAL_WAITING_SYNC state, the finger waits for the SYNC_10M signal, which is active low. When the falling edge of SYNC_10M is detected, the frame counter F_CNT is set to zero, and the state changes to SYNC_DETECTED. In the SYNC_DETECTED state, the frame counter starts counting CHIPX8 cycles and if SYNC_10M goes to high, the state is changed to WAITING_SYNC. In the WAINTING_SYNC state, F_CNT continues counting CHIPX8 cycles and waits for the next SYNC_10M's falling edge. When the finger detects the falling edge of SYNC_10M again, F_CNT is reset to zero, and the state is changed to SYNC_DETECTED. This process repeats.

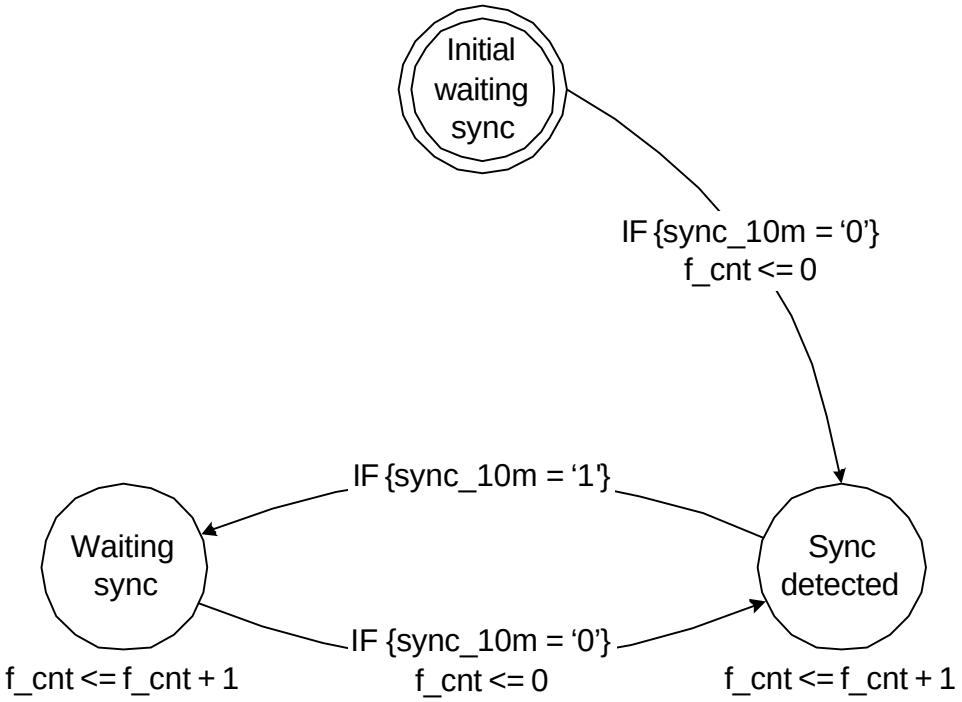


Figure 4.6: Frame state machine

4.2.1.2.3 Reset state machine

The *reset state machine* controls the RESET_SYS signal through the START signal, which is START_USER & FINGER_LOCK, where & denotes the AND operation. Thus, a START signal of '1' implies that the finger is operating.

RESET_SYS is RESET & RESET_SIG, where RESET is the external reset signal and RESET_SIG is the internal reset signal, which is handled by the *reset state machine*.

Figure 4.7 shows the *reset state machine*. This state machine starts from the START_DEACTIVATED state. In this state the internal signal RESET_SIG is set to 1 to make the system reset signal, RESET_SYS, 1. It means that the system is ready to operate as soon as START changes to 1. If START changes to 1, the state changes to START_ACTIVATED, and the finger is ready to operate. When START changes to 0, which means that the mobile is turned off or the cell searcher has lost the connection with the base station, RESET_SIG is set to 0 and thus RESET_SYS is also set to 0. As a result, the whole finger operation is reset and the state changes to START_DEACTIVATED.

An individual finger can lose the link with the base station irregularly even though the mobile has open traffic, because the multipath fading environment causes a different path condition for each finger. If a finger has lost the connection to the base station, it is said that the finger is unlocked. Even in the case that a finger's condition changes from unlock to lock in the middle of a frame, the finger's operation should start at the beginning of a frame. This is because a finger's internal operation including the code generations has a period of one frame. Therefore, we generate the RESET_SYS signal to reset the internal blocks of the fingers before the start of a frame.

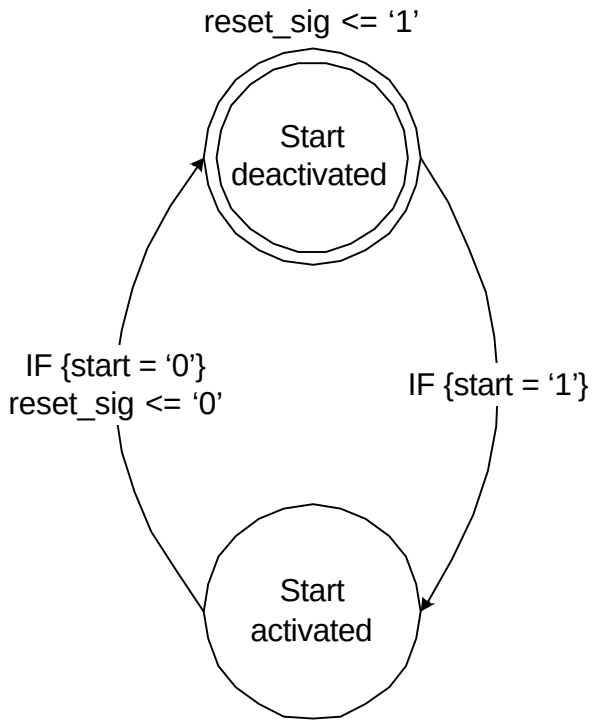


Figure 4.7: Reset state machine

4.2.1.2.4 Frame edge state machine

The *Frame edge state machine* generates FRAME_EDGE_CODE and START_CODE_GEN signals at the starting point of a frame to signal the code generators to start making the proper codes in advance of one CHIPX1 cycle of the demodulator's operation.

Figure 4.8 depicts the *frame edge state machine*. It starts with the INITIAL state. When the *operation state machine* finishes the count of the proper time offset value, FRAME_EDGE_CODE is set to 1 to report to the code generator the start of a new frame and make the code generator start the code generation with a new seed value. If the coming frame is the first frame for a finger, START_CODE_GEN is set to high to enable the code generation. Then, the state is changed to HOLDING for a CHIPX1 clock signal to be able to see the FRAME_EDGE_CODE signal's status. In the next CHIPX8 cycle, the state is changed to FRAME_EDGE_RESET and FRAME_EDGE_CODE is set back to 0. Then, the state is goes back to INITIAL. This process repeats.

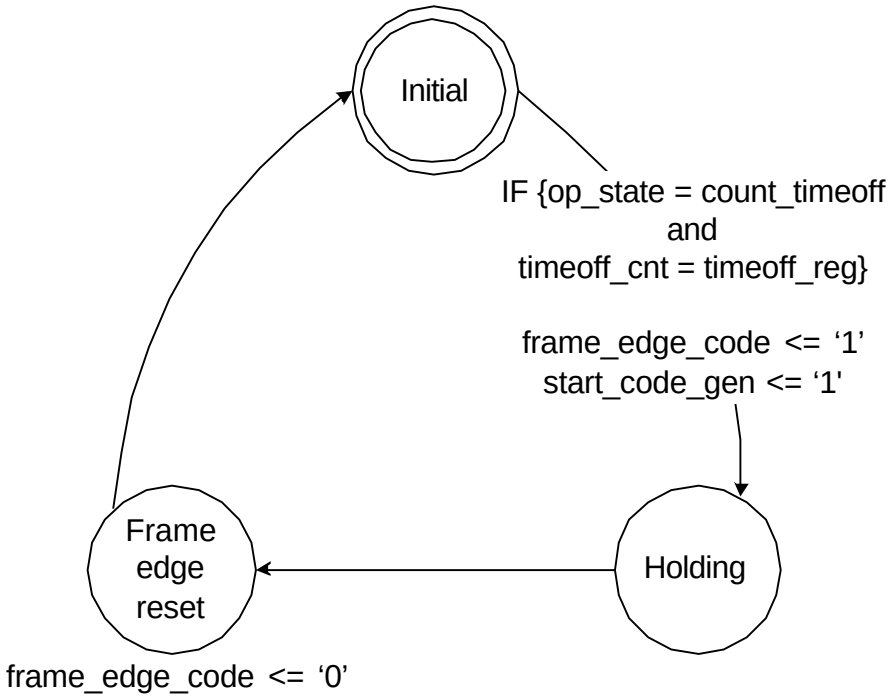


Figure 4.8: Frame edge state machine

4.2.1.2.5 Chipx1 state machine

The *Chipx1 state machine* generates the chip clock signal CHIPX1 from the 1/8-chip clock signal CHIPX8 for the finger operation. Figure 4.9 shows the *chipx1 state machine*. In the INITIAL state, when the *operation state machine* indicates the timing for the start of the clock generation, the chip timing clock signal CHIPX1 is generated and the CHIPX8 counter CLK_CNT is reset. The state changes to FREE_RUN. Once the state changes to FREE_RUN, the CHIPX1 clock is generated; when the operation state is CLK_START or every eight CHIPX8 cycles if the chipx1 count state is not HOLD.

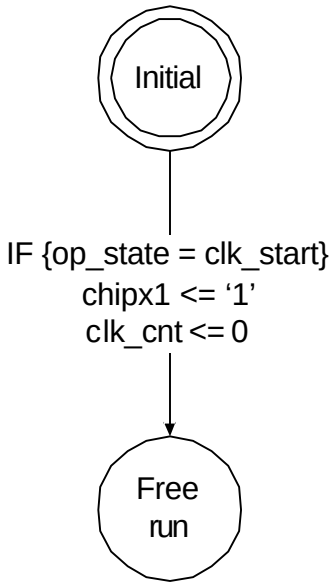


Figure 4.9: Chipx1 state machine

4.2.1.2.6 Chipx1 count state machine

The *Chipx1 count state machine* is used to prevent the CHIPX1 clock from exceeding the maximum value of 38400 cycles within a frame through counting the number of generated CHIPX1 cycles. The accidental generation of a CHIPX1 cycle can be caused mainly by two reasons. One is inaccurate local oscillator frequency and the other is imperfect synchronization information from the synchronization channel. The accidental CHIPX1 clocks can ruin the finger's internal operations, thus the number of generated CHIPX1 cycles is counted.

Figure 4.10 shows the *chipx1 count state machine*. The state is changed from INITIAL to FREE_RUN at the first CHIPX1 clock generation, and the chipx1 counter does not count this cycle, because the first CHIPX1 clock is for the code generators in advance of the data frame start. In the FREE_RUN state, the CHIPX1 counter counts every CHIPX1 cycle, and if the counter value is 38399, which is the maximum number of CHIPX1 cycles during a frame, the state changes to HOLD. In the HOLD state, the CHIPX1 generator does not generate CHIPX1 clock signal but waits until the operation state indicates the CHIPX1 generation for the new frame, which is CLK_START of the operation state. At that moment, the chipx1 count state is changed to FREE_RUN again

and CHIPX1_CNT is set to the largest value, which is all 1's, so that in the FREE_RUN state CHIPX1_CNT can start from the value 0 for the new frame.

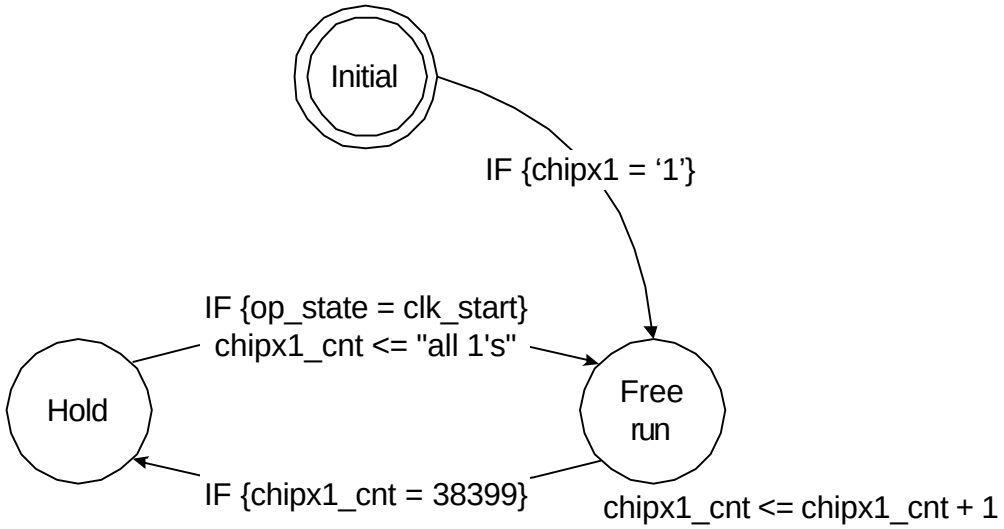


Figure 4.10: Chipx1 count state machine

4.2.1.3 Calculation of the time offset

The time-offset information, which is loaded when the operation state is LOAD_TIMEOFF, is used for the calibration of the time-offset value in the CALC_TIMEOFF state. The calculation of the time-offset value is different for varying situations. There are two control parameters for the calculation options. One is TIMEOFF_FAIL_RECOVER and the other is TRACK_OFF.

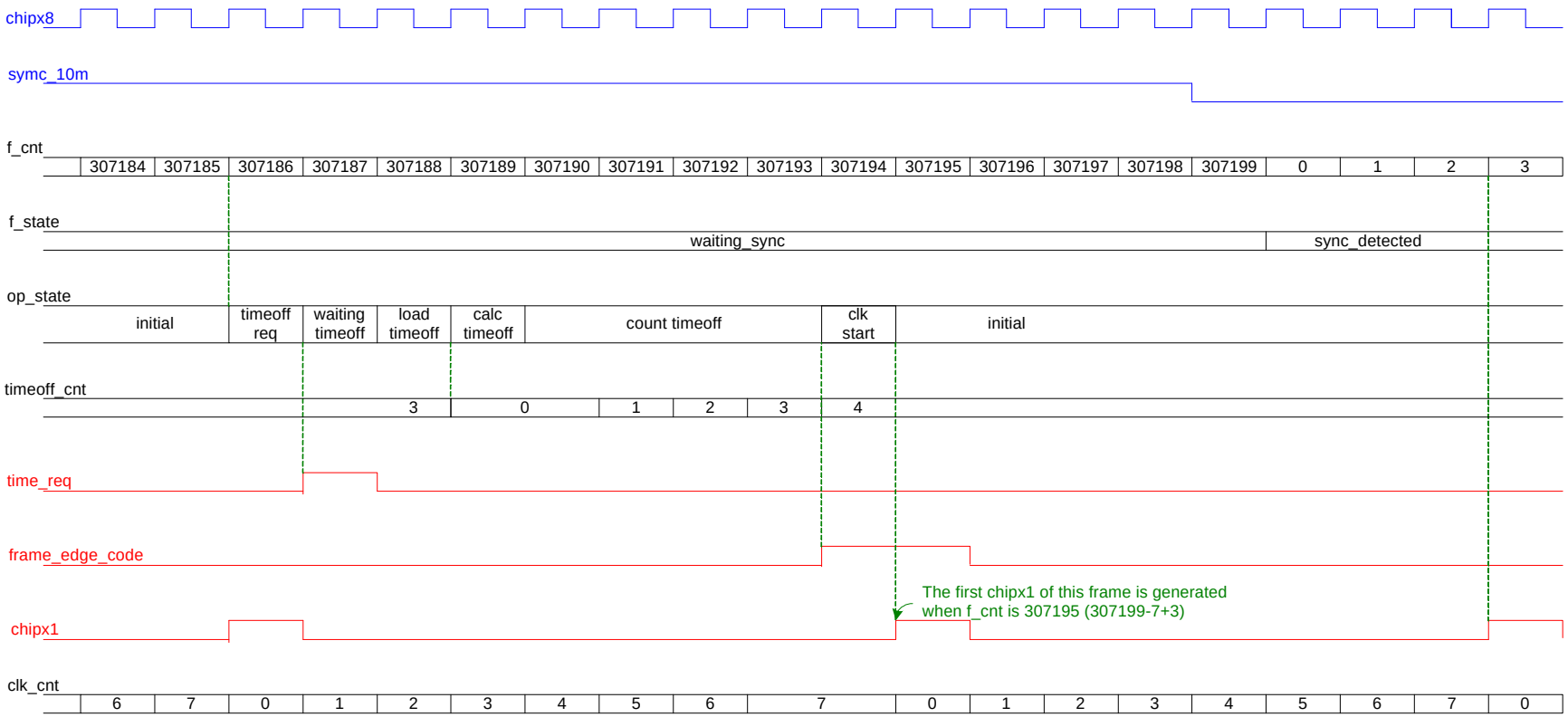
When the calculated time offset value is larger than 38400*8 or less than zero, which results in that the calculated current frame's starting point is over the boundary of current 10 msec synchronization period, we say time offset adjustment has failed. When time offset adjustment has failed, we have two choices according to the user defined setting TIMEOFF_FAIL_RECOVER. When the recover flag is on, the time offset value is recovered to the time offset value received from the cell searcher, ignoring the result from the time tracker, but when the recover flag is off, the time offset value follows the previous time offset, which is adopted for the previous frame. The recover flag

TIMEOFF_FAIL_RECOVER is user defined, thus can be changed by the higher layer's decision.

If the time offset given by the cell searcher is changed, the time tracker output will not acknowledge the time offset by setting the track flag TRACK_OFF as on, because when the major time offset is changed the fine tracking result is no longer valid. The track off flag will be set to low to adopt the time tracking result if the time offset value currently received from the cell searcher is the same as the previous value.

4.2.1.4 Timing diagram

Figure 4.11 shows an example waveform of the clock generator. We assume that the received SYNC_10M is the third one, thus RESET_SYS and START_CODE_GEN have already been generated when the first SYNC_10M is received. START_USER and FINGER_LOCK are also assumed to be 1's, which means that the finger is locked by cell searcher and properly working. In this example waveform, the time-offset information TIMEOFF_SC is 2 and the time tracking result LEAD_LAG is lag. Thus the clock signal CHIPX1 should be generated when the frame counter F_CNT is 3, and this results in the generation of the first CHIPX1 of this frame occuring when the F_CNT is changed to 307195, which is 307199-7+3.



Note
1. time offset condition: timeoff_sc = 2 and lead_lag = lag
2. start_user = ‘1’ and finger_lock = ‘1’

Figure 4.11: Waveform of the clock generator

4.2.2 Control generator (CTRL_GEN)

The control generator generates the control signals for the entire finger block, except the code generators referring to the signals generated by the clock generator for the finger operations. Figure 4.12 and Table 4.2 shows the input and output signals for the control generator.

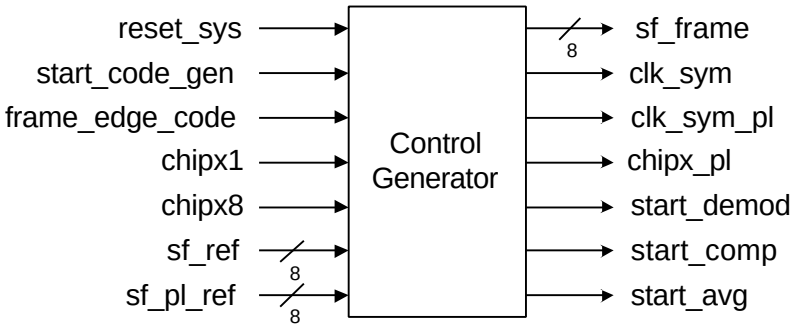


Figure 4.12: Control generator

Table 4.2: I/O signals in the control generator

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset signal for the inner system of the finger
Start_code_gen	I	1	Enable signal for code generator
Frame_edge_code	I	1	Frame edge for the code generators. It is considered as an indicator of the initialization to be prepared for the next frame.
Chipx1	I	1	Chip clock (3.84 MHz)
Chipx8	I	1	1/8 chip clock (30.72 MHz)
Sf_ref	I	8	Spreading factor for the DPCH
Sf_pl_ref	I	8	Spreading factor for the CPICH
Sf_frame	O	8	Sends DPCH spreading factor information to the deskew-combiner
Clk_sym	O	1	Enable signal for the load of the DPCH symbol for the following blocks after the demodulation
Clk_sym_pl	O	1	Enable signal for the load of the CPICH symbol for the following blocks after the demodulation
Chipx_pl	O	1	Clock signal for the load of the moving-averaged CPICH symbol for CF estimator, power estimator and frequency offset estimator (15 KHz)
Start_demod	O	1	Enable signal for the demodulator
Start_comp	O	1	Enable signal for the compensator
Start_avg	O	1	Enable signal for the moving average

After the reset of the entire finger's individual blocks by RESET_SYS, START_CODE_GEN informs the start of the code generation. This is the first control signal coming into the finger, thus other control signals that indicate the start of any operations will follow START_CODE_GEN. At that moment, which is nine CHIPX8 cycles ahead of the frame starting point, FRAME_EDGE_CODE initializes the finger by loading the spreading factor for the DPCH and CPICH.

When the initialization is finished, the control generator generates SF_FRAME to give the DPCH spreading factor information for the deskew-combiner. Let us assume that the time-offset is zero. On the first rising edge of CHIPX1, if the code generator is enabled, which means that START_COED_GEN is '1', START_DEMOD is also set to '1' so that the demodulator can read the enable signal START_DEMOD and know that demodulation is ready for the first CHIPX1 cycle after the falling edge of synchronization signal. At the next CHIPX1 cycle, START_COMP is set to '1' if the demodulator is enabled and the first chip is demodulated, then at the next CLK_SYM, the loading of the first DPCH symbol is possible in the compensator. START_AVG is also generated at the same time so that at the next CLK_SYM_PL the loading of the CPICH symbol can be performed in the moving average block.

The CLK_SYM signal is generated at the end of a DPCH symbol to notify that demodulation for one DPCH symbol is finished. Also, at the end of a CPICH symbol, CLK_SYM_PL is generated to notify that one pilot symbol is generated. After the generation of CLK_SYM_PL, a clock signal CHIPX_PL is generated for the load of the moving averaged CPICH symbols in the CF estimator, frequency offset estimator and power estimator. The waveform for the whole process of the control generator is given in Figure 4.13 regarding the starting part of a frame and Figure 4.14 regarding the first CPICH symbol's generation part.

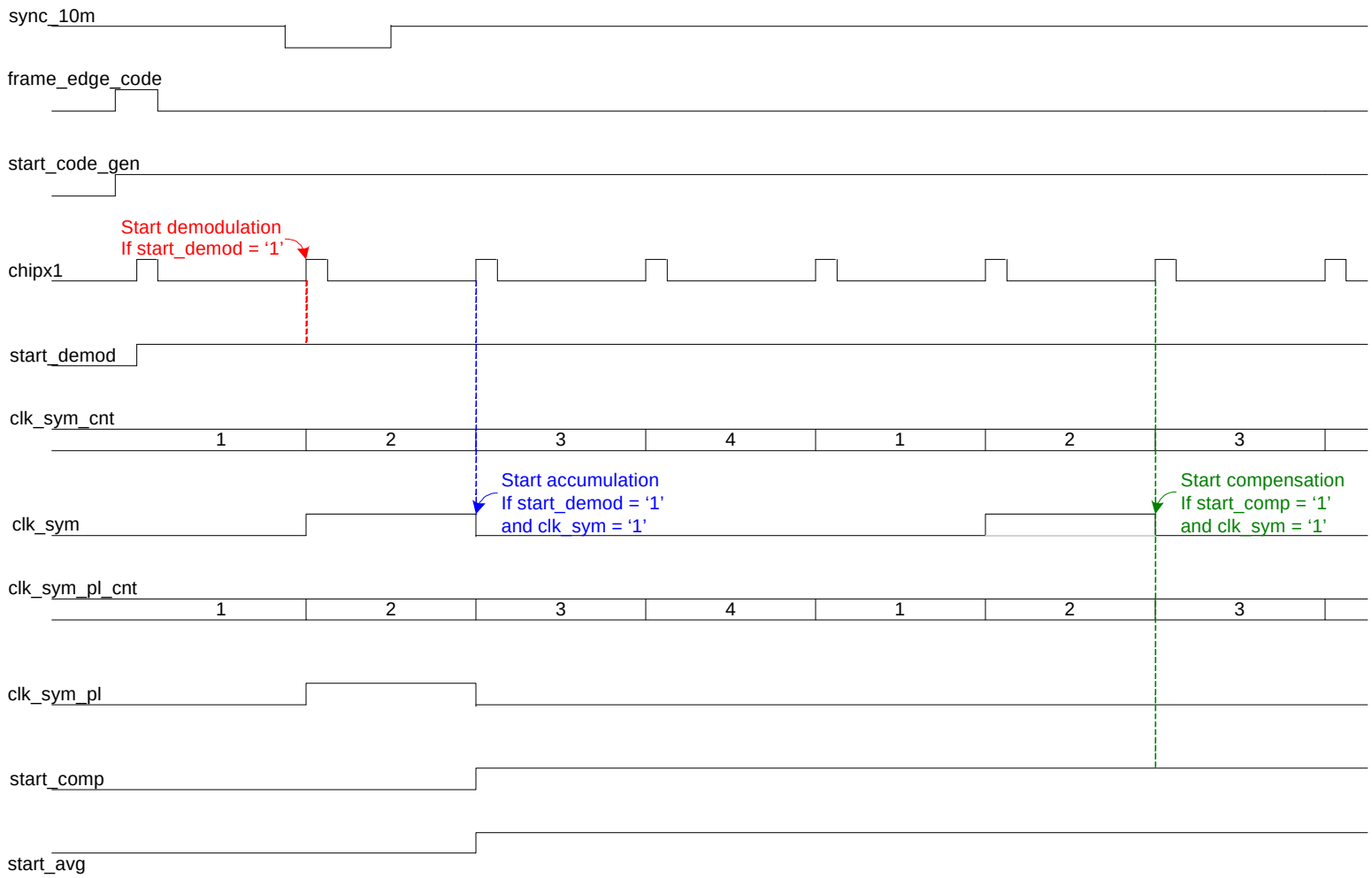


Figure 4.13: Waveform of the control generator that shows the starting part of a frame. DPCH spreading factor is 4. The control signals clk_sym, clk_sym_pl, start_demod, start_comp and start_avg are generated on the rising edge of the clock signal chipx1.

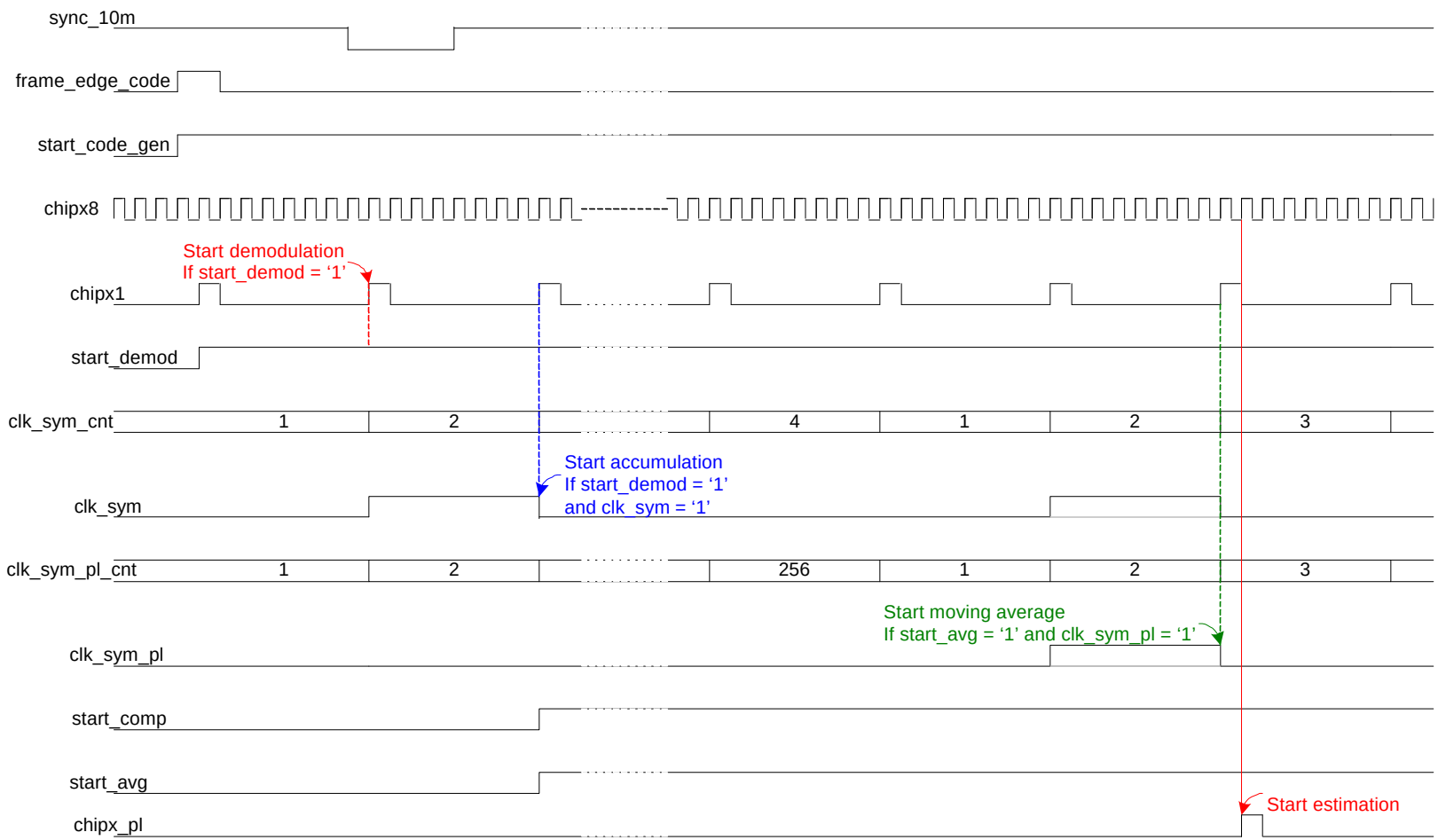


Figure 4.14: Waveform of the control generator that shows the starting part of a frame and the ending part of the first CPICH symbol. DPCH spreading factor is 4. The clock signal for the load of moving averaged CPICH symbol in the estimators, cihpx_pl, is generated on the rising edge of the clock signal chipx8.

4.2.3 Demodulator (DEMOD)

A demodulator performs the despreading operations. Only on-time chips are demodulated for the recovery of the desired symbol of the DPCH. However, early and late samples are also demodulated as well as the on-time samples for the CPICH. Thus, as we can see in Figure 4.3, a finger needs four copies of the demodulator.

The inputs to the demodulator are the received chip sequence, the OVSF code, and the scrambling code. Figure 4.15 shows the demodulator for one channel. S_I and S_Q represent the real and imaginary part of the scrambling code, and C denotes the OVSF code.

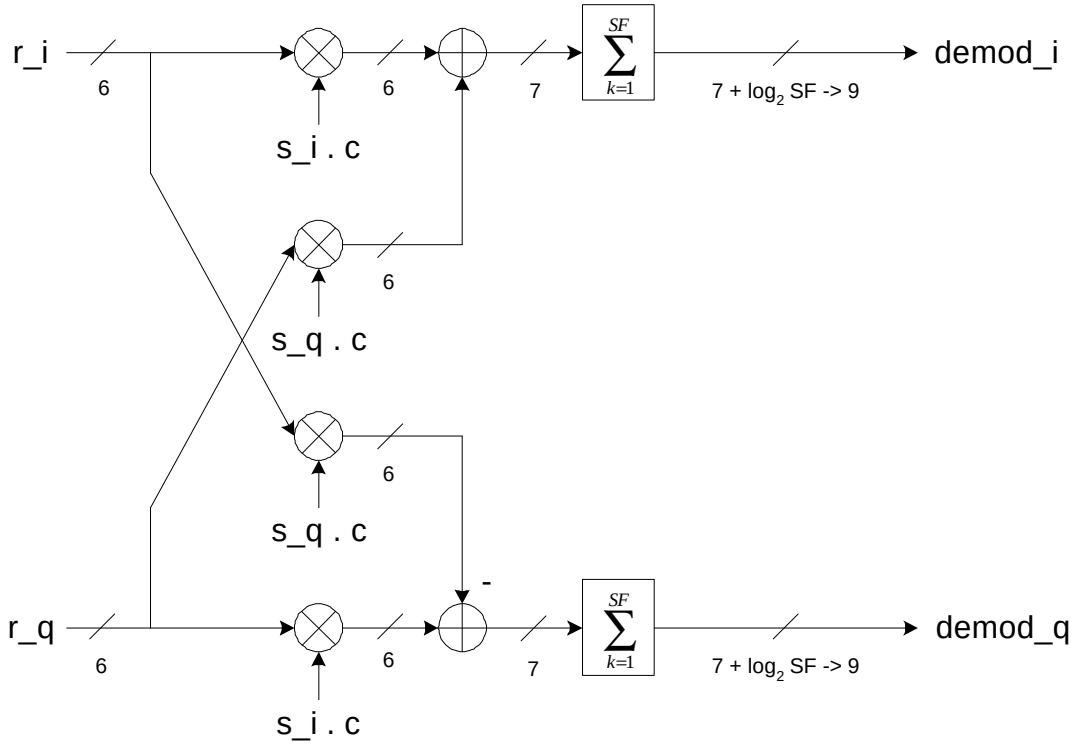


Figure 4.15: Demodulator

The demodulator uses the CHIPX1 clock for the operation. Therefore, it receives the chip sequences every CHIPX1 and outputs the symbols at every CLK_SYM (or CLK_SYM_PL). The I/O signals are described in Table 4.3.

Table 4.3: I/O signals in the demodulator

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Start_demod	I	1	Enable signal for the demodulator
Frame_dege_code	I	1	Frame edge for the code generators
Chipx1	I	1	Chip clock (3.84 MHz)
Clk_sym (clk_sym_pl)	I	1	Enable signal for the load of every DPCH (CPICH) symbol
Sf_ref	I	8	Spreading factor
R_i	I	6	I channel chip
R_q	I	6	Q channel chip
S_i	I	1	Real part of scrambling code
S_q	I	1	Imaginary part of scrambling code
C	I	1	OVSF code
Demod_I	O	9	I channel symbol
Demod_q	O	9	Q channel symbol

4.2.3.1 Output bit truncation

As we can see in Figure 4.15, the output bits are truncated to 9 bits. Without the bit truncation of the output, the outputs of DPCH demodulation have variable length from 9 bits to 16 bits according to the spreading factor, which vaies from 4 to 512. The demodulator outputs of CPICH have 15 bits. The easiest way is to use the maximum length of 16 bits for the outputs, at the cost of a large die area. Since DEMOD_I and DEMOD_Q are used for the computations and estimations in the following operational blocks, we truncate the output bits as small as possible without any serious performance loss.

Figures 4.16 to 4.23 show the performance comparison with the different bit truncation for all spreading factors. They provide the performance loss in dB scale compared with the performance without truncation. Let us consider the case for the spreading factor of 512 shown in Figure 4.23, because it needs the larges bit truncation and thus incurs the largest performance loss. The maximum performance loss due to the truncation is 1.0302 dB for the 8-bit output case and the average performance loss is 0.8312 dB, while the 9-bit truncation incurs the maximum 0.3690 dB and average 0.2513 dB of performance loss. The increase of the circuit area from 8 bits to 9 bits is significant. However, 9-bit output achieves 30.23% better performance than 8-bit output. Thus, the

output length for the proposed demodulator is truncated to 9 bits. Note that 9 bits are sufficient for the spreading factor 4. The performances of 8-bit and 9-bit outputs are tabulated in Table 4.4.

Table 4.4: Performance degradation with differing output bits for the demodulator

Spreading Factor	Performance loss (dB)			
	8-bit output		9-bit output	
	Average	Maximum	Average	Maximum
4	0.0689	0.1303	0	0
8	0.0755	0.1642	0.0256	0.0930
16	0.0586	0.1178	-0.0056	0.0471
32	0.1394	0.2567	-0.0079	0.1369
64	0.1603	0.3354	0.0203	0.1957
128	0.2145	0.2527	0.0274	0.0981
256	0.3638	0.4266	0.0870	0.1124
512	0.8312	1.0302	0.2513	0.3690

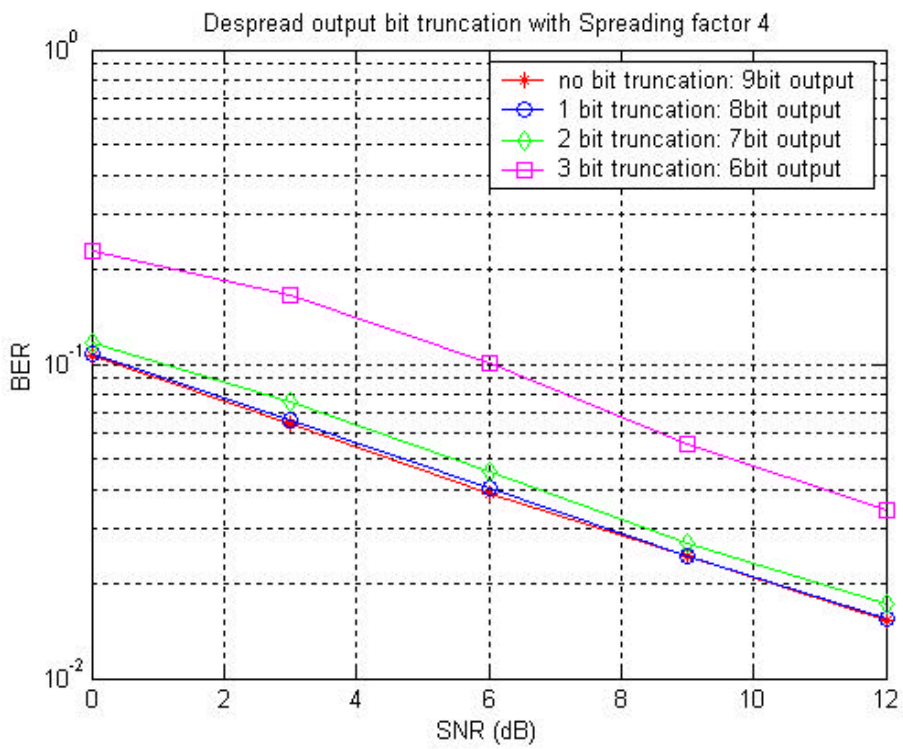


Figure 4.16: Bit truncation of demodulation output with spreading factor 4

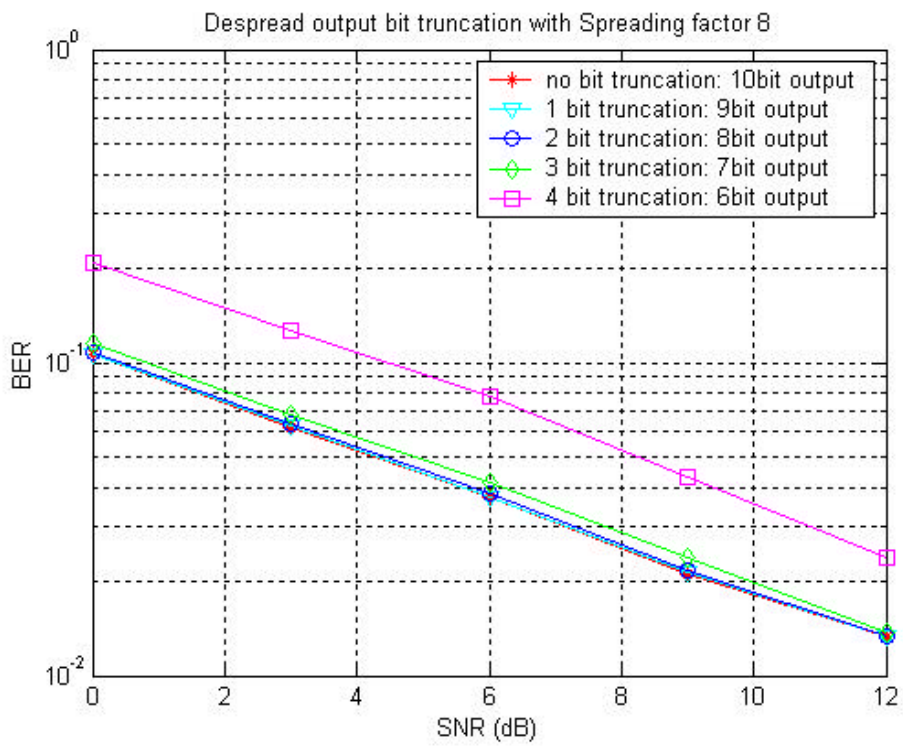


Figure 4.17: Bit truncation of demodulation output with spreading factor 8

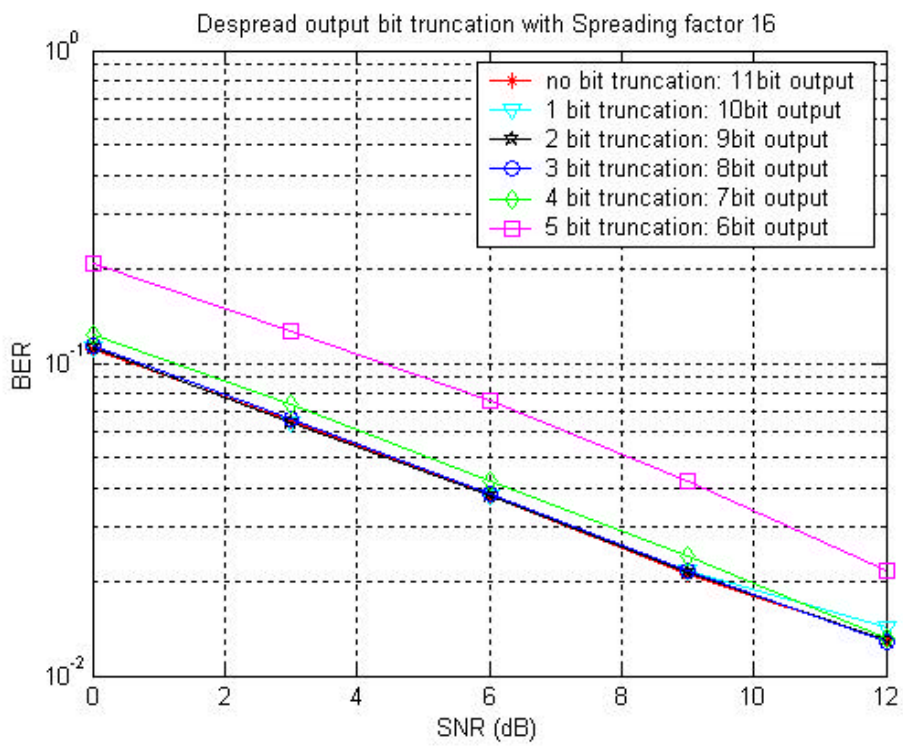


Figure 4.18: Bit truncation of demodulation output with spreading factor 16

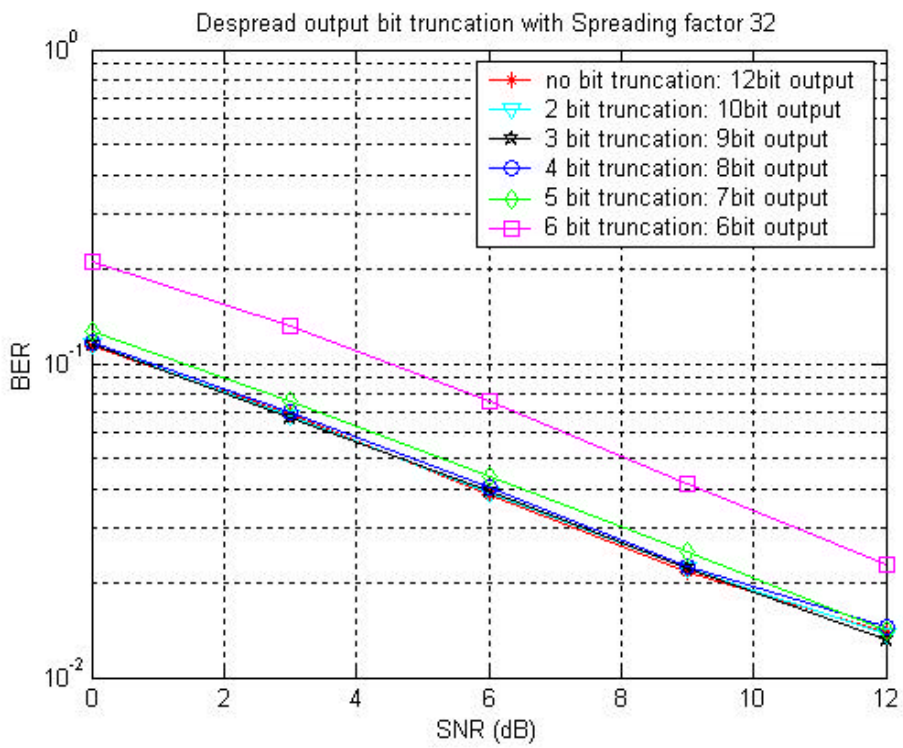


Figure 4.19: Bit truncation of demodulation output with spreading factor 32

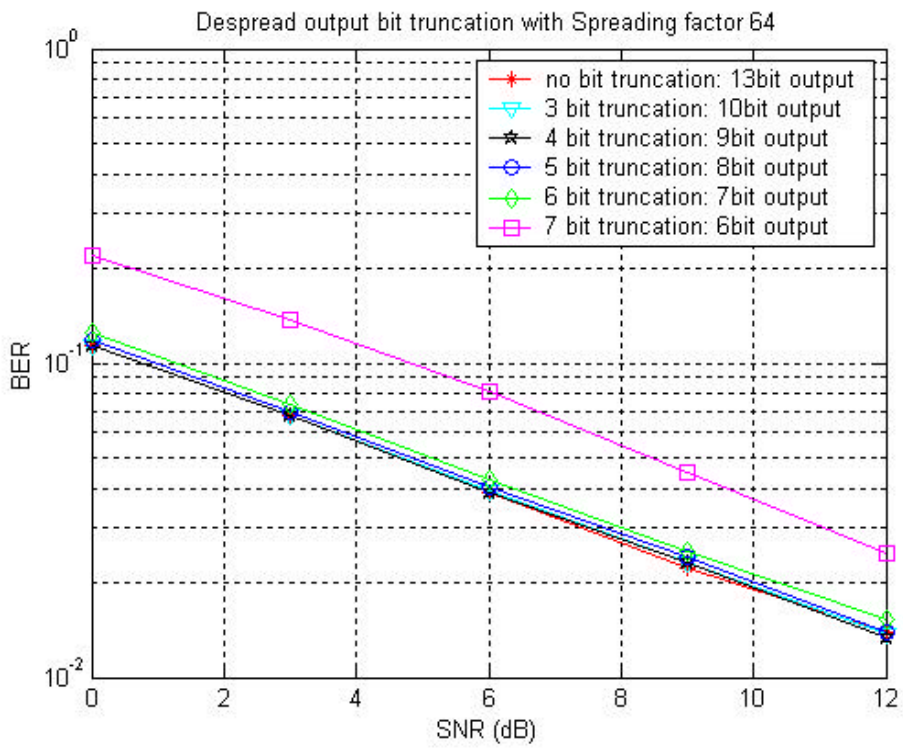


Figure 4.20: Bit truncation of demodulation output with spreading factor 64

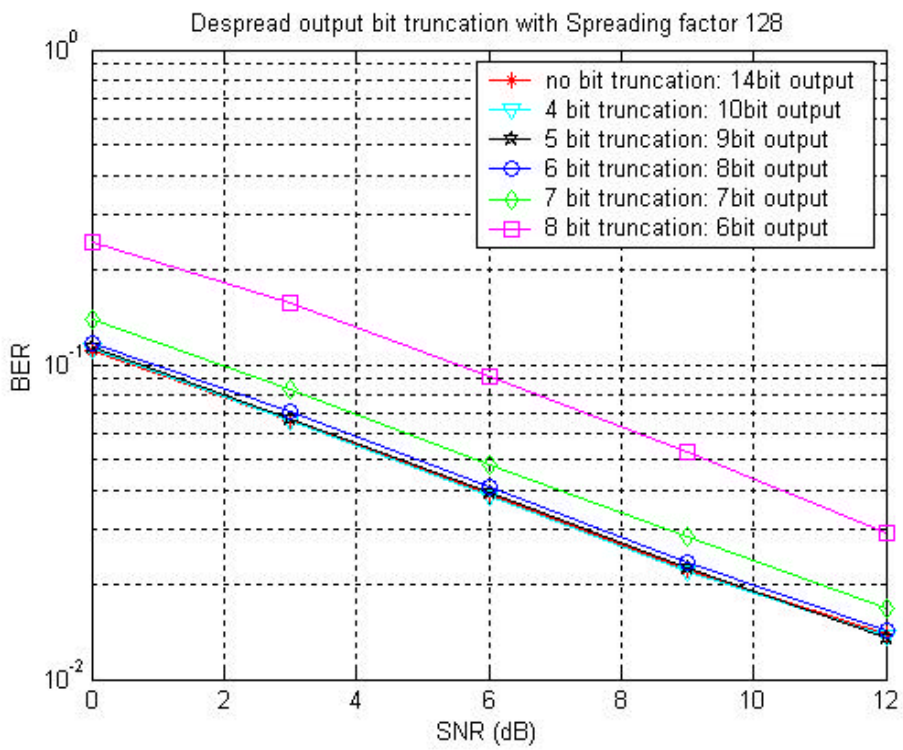


Figure 4.21: Bit truncation of demodulation output with spreading factor 128

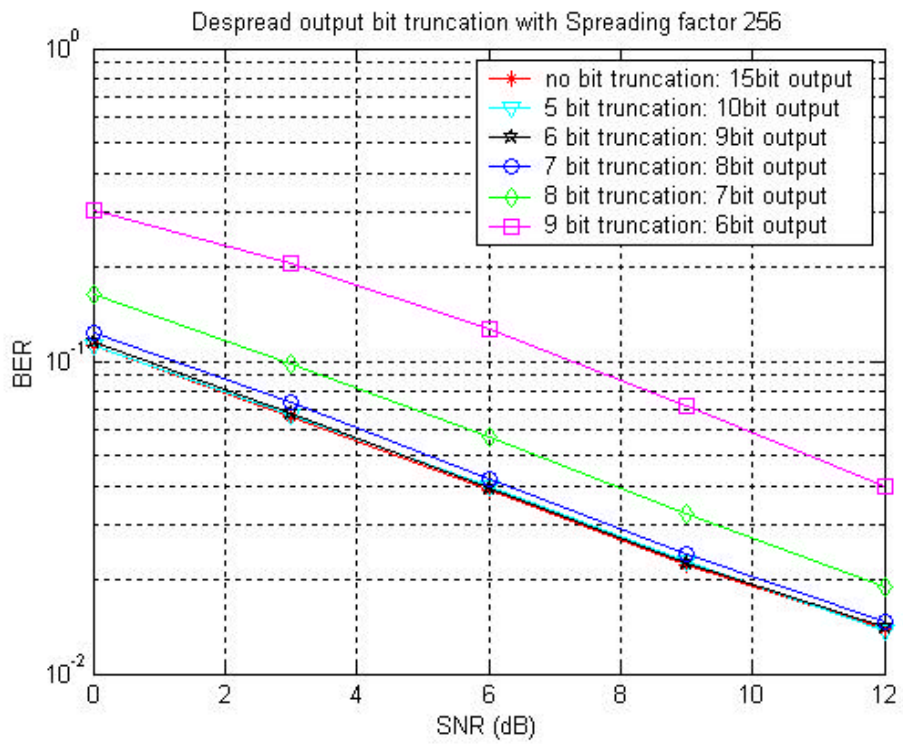


Figure 4.22: Bit truncation of demodulation output with spreading factor 256

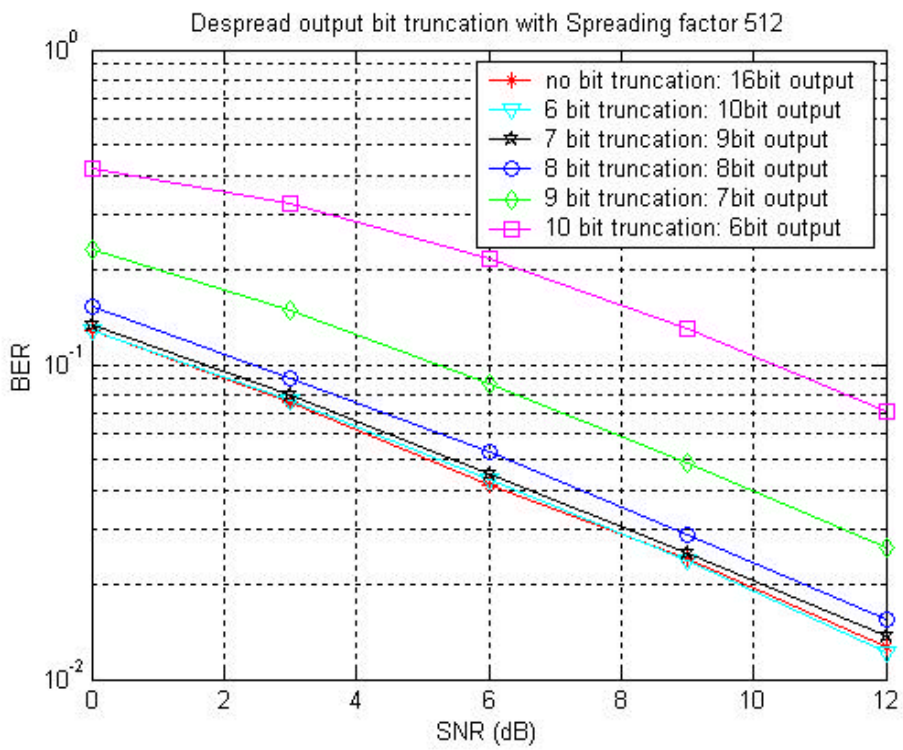


Figure 4.23: Bit truncation of demodulation output with spreading factor 512

4.2.4 Moving average block (AVG)

The moving-average of CPICH symbols are used for the CF estimator, frequency offset estimator and power estimator. Since the moving average length is determined as 4 CPICH symbols, the accumulated values are 2-bit right shifted. Figure 4.24 shows the block diagram of the moving average block. This block receives the CPICH symbol at every CPICH symbol enable signal, CLK_SYM_PL, and outputs the moving averaged value. The I/O signals are described in Table 4.5.

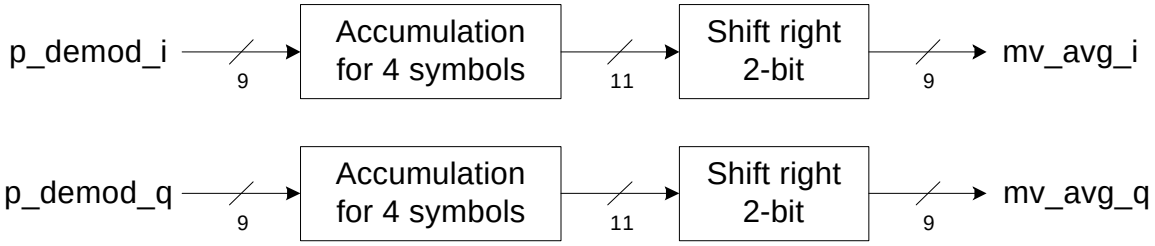


Figure 4.24: Moving average block

Table 4.5: I/O signals in the moving average block

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Start_avg	I	1	Enable signal for the moving average
Chipx1	I	1	Chip clock (3.84 MHz)
Clk_sym_pl	I	1	Enable signal for the load of every CPICH symbol
Pl_demod_I	I	9	I channel CPICH symbol
Pl_demod_q	I	9	Q channel CPICH symbol
Mv_avg_I	O	9	Moving averaged I channel CPICH symbol
Mv_avg_q	O	9	Moving averaged I channel CPICH symbol

4.2.5 CF estimator (CF_EST)

The CF estimation is performed with moving-averaged CPICH symbols. Figure 4.25 shows the structure of the CF estimator. Table 4.6 shows the input and output signal description for the CF estimator. The CF estimator receives the moving averaged CPICH symbols as inputs at every CHIPX_PL, which is one CHIPX8 cycle after CLK_SYM_PL, and outputs the estimated cosine and sine values.

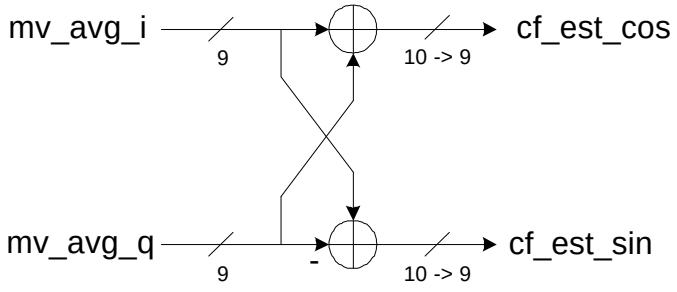


Figure 4.25: Channel / frequency offset Estimator

Table 4.6: I/O signals in channel / frequency offset estimator

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Chipx_pl	I	1	Clock signal for the load of the moving-averaged CPICH symbol (15 KHz)
Mv_avg_I	I	9	Moving averaged CPICH I channel symbol
Mv_avg_q	I	9	Moving averaged CPICH Q channel symbol
Cf_est_cos	O	9	Cos value of channel estimation result
Cf_est_sin	O	9	Sin value of channel estimation result

4.2.5.1 Output bit truncation

The CF estimator output can also be truncated. Figure 4.26 shows the performance with the different output bits of the CF estimator. The outputs can be truncated to 9 bits with a performance loss average of 0.1309 dB and a maximum of 0.1562 dB. The performance is measured with the spreading factor of eight, and the performance loss includes one caused by the bit truncation of the demodulator outputs.

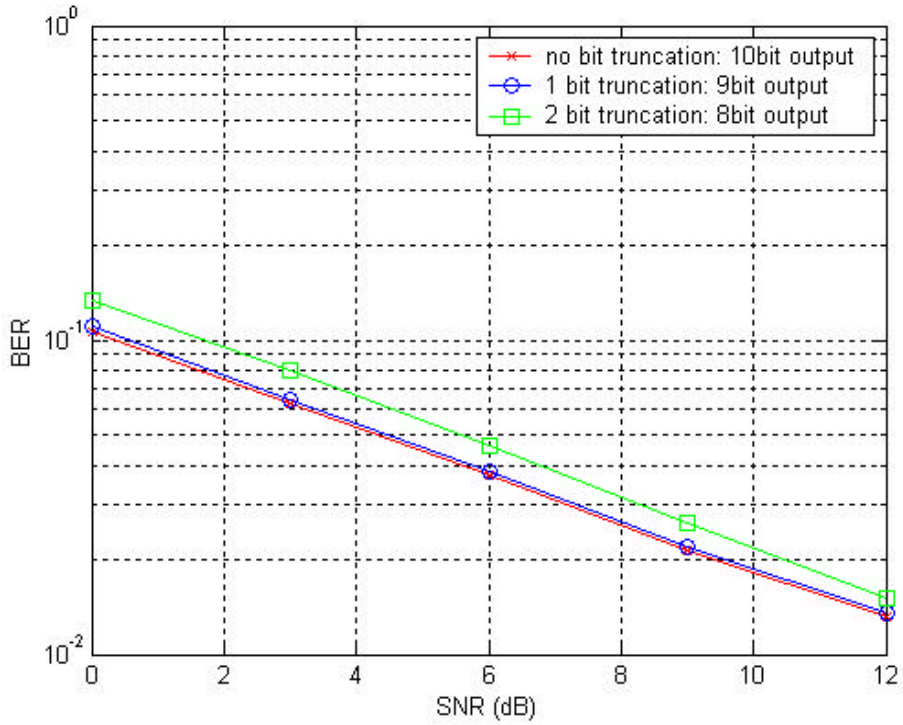


Figure 4.26: Bit truncation for CF estimator output with 9-bit demodulation output

4.2.6 Compensator (COMP)

Compensation is applied to DPCH symbols. Figure 4.27 shows the block diagram of the compensator. I/O signals are described in Table 4.7. Since the first CF estimation result is available after a 256-chip period from the frame start, which is one CPICH symbol period, the first 256/SF DPCH symbol sequences are not compensated. For example, if the DPCH spreading factor is 4, the first 64 symbols are not compensated. The first symbol of the next CPICH symbol period can be compensated with the CF result calculated from the last CPICH symbols. When the compensation is finished, the control signal DATA_AV is sent to the deskew-combiner.

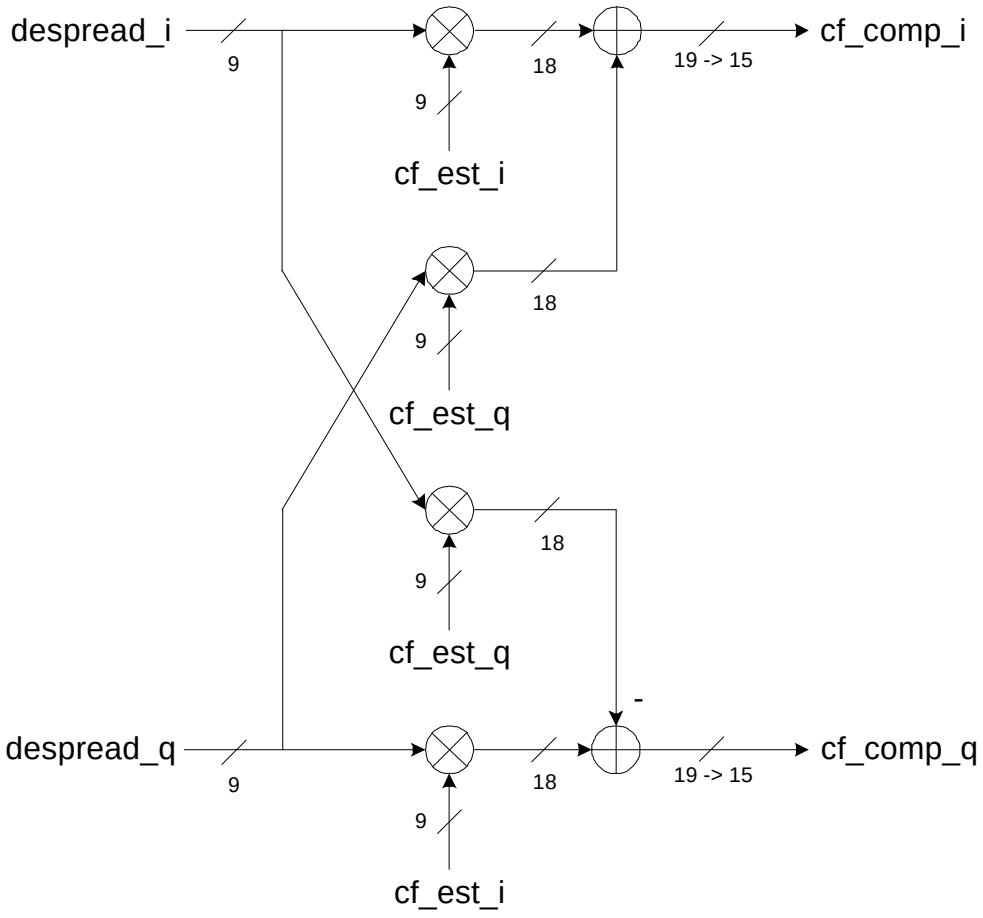


Figure 4.27: Compensator

Table 4.7: I/O signals in channel / frequency offset compensator

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Start_comp	I	1	Enable signal for the compensator
Chipx1	I	1	Chip clock (3.84 MHz)
Clk_sym	I	1	Enable signal for the load of the DPCH symbol
Demod_I	I	9	I channel symbol
Demod_q	I	9	Q channel symbol
Cf_est_cos	I	9	Cos value of the CF estimation result
Cf_est_sin	I	9	Sin value of the CF estimation result
Comp_I	O	15	Compensated I channel symbol
Comp_q	O	15	Compensated Q channel symbol
Data_av	O	1	Indicates that the data is available, goes to deskew-combiner

4.2.6.1 Output bit truncation

Figure 4.28 provides the performance with different numbers of truncated bits. 19 bit outputs are truncated to 15 bits with the performance loss average of 0.2341 dB and maximum of 0.2797 dB. The performance is measured with the spreading factor of eight, and the performance loss includes ones caused by the bit truncations of the demodulator outputs and CF estimator outputs.

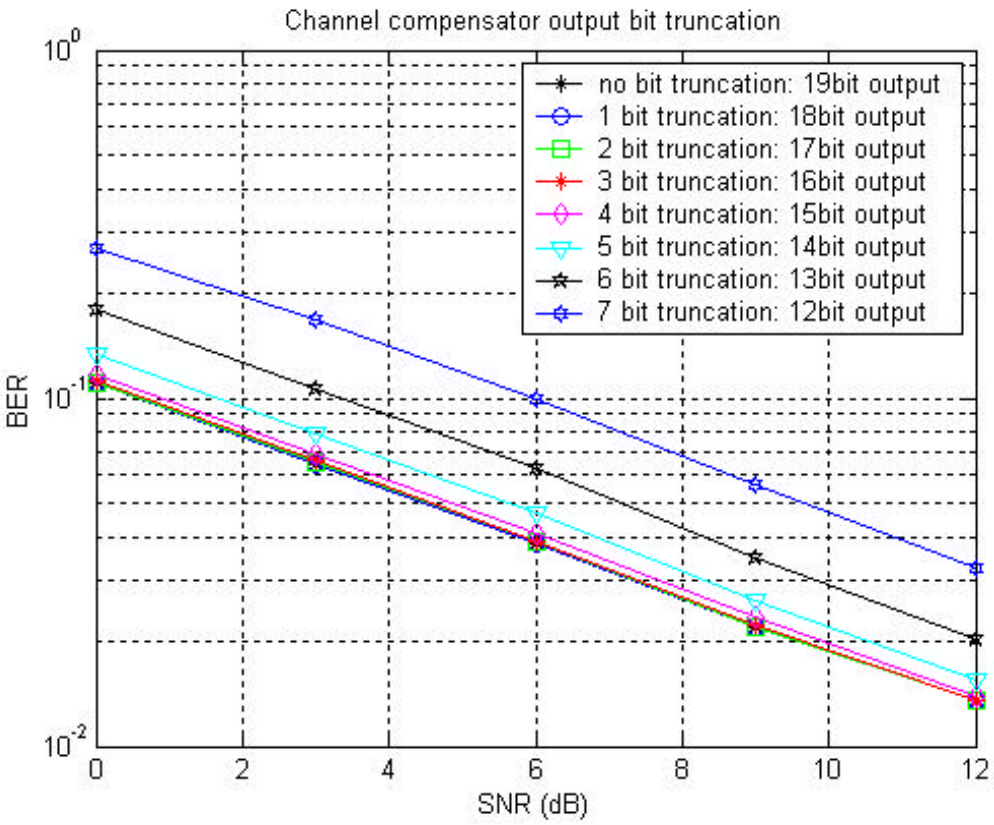


Figure 4.28: Bit truncation of channel / frequency offset compensator

4.2.7 Frequency offset estimator (FREQ_EST)

The frequency offset estimator and frequency offset compensator can be eliminated as described in Chapter 3, because the frequency offset can be estimated and compensated along with the channel phase rotation through the channel estimator and channel compensator. However, the frequency-offset estimator is necessary to provide the frequency-offset information to the DSP. Though the shift of division operation and the provision of a cosine/sine lookup table on the DSP side, the frequency offset estimator contains only the blocks before the division block. The resulting frequency offset estimator is shown in Figure 4.29.

The accumulator at the end of the frequency-offset estimator should not be moved to the DSP in an attempt to lower the frequency of the interrupt. This is because a fast interrupt causes a processing burden for the DSP. The frequency of the interrupt should

be maintained as low as possible, and the interrupt is generated once a frame, which is a 10-msec period.

The moving average blocks shown in Figure 4.24 are shared with the CF estimator. Thus, the inputs to the frequency-offset estimator are moving-averaged CPICH symbols, which are for the CF estimator. The I/O signal descriptions are given in Table 4.8. The full output bits, which are 27, are used without truncation; because the cosine and sine values of the estimated frequency offset are very small, the bit truncation can cause a significant error in the resulting frequency offset value.

Table 4.8: I/O signals in frequency offset estimator

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Chipx_pl	I	1	Clock signal for the load of the moving-averaged CPICH symbol (15 KHz)
Chipx8	I	1	1/8 chip clock (30.72 MHz)
Mv_avg_I	I	9	Moving averaged CPICH I channel symbol
Mv_avg_q	I	9	Moving averaged CPICH Q channel symbol
Freq_est_cos	O	27	Cos value of frequency offset estimation result
Freq_est_cos	O	27	Sin value of frequency offset estimation result
Freq_est_av	O	1	Indicates that the frequency-offset values are available. Goes to the interrupt controller

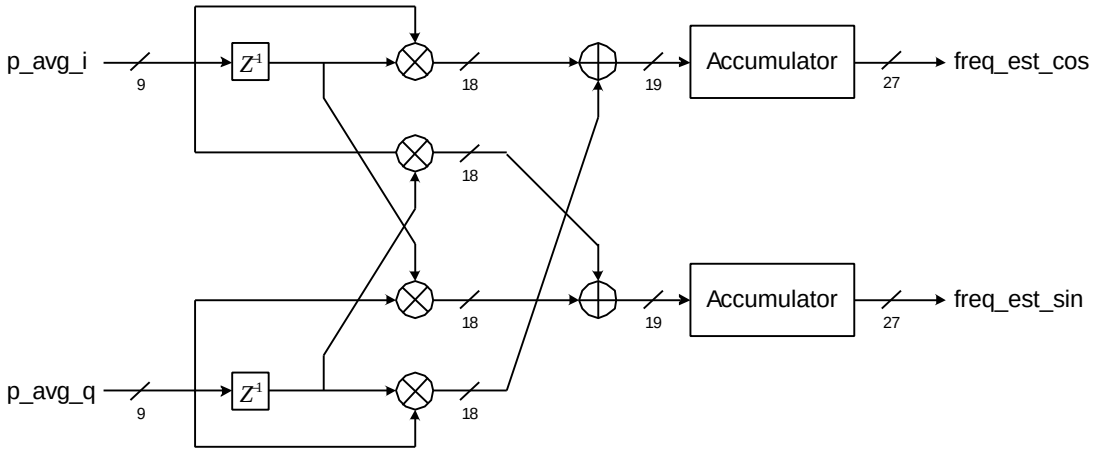


Figure 4.29: Frequency offset estimator to send the estimated value to the DSP

4.2.8 Power Estimator (POW_EST)

The estimated power is sent to the finger on/off decision block and the DSP. The finger on/off decision block receives the estimated power level at every CPICH symbol to follow the channel variation and the DSP receives the estimated power at every frame for the control of the transmit power.

As explained in Section 3.2.8, the approximation method, which calculates the magnitude value, is used for the power measurement. Figure 4.30 shows the structure of the power estimator. The I/O signals are described in Table 4.9.

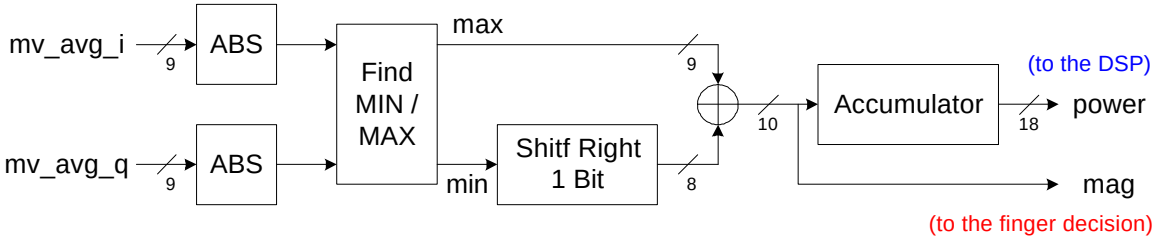


Figure 4.30: Power estimator

Table 4.9: I/O signals in power estimator

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Chipx_pl	I	1	Clock signal for the load of the moving-averaged CPICH symbol (15 KHz)
Chipx8	I	1	1/8 chip clock (30.72 MHz)
Mv_avg_I	I	9	Moving averaged CPICH I channel symbol
Mv_avg_q	I	9	Moving averaged CPICH Q channel symbol
Mag	O	10	Estimated power level for the finger decision block
Power	O	18	Estimated power level for the DSP
Power_av	O	1	Indicates that the power estimation value is available. Goes to the interrupt controller

4.2.9 Time Tracker (TIME_TRK)

The structure of the finger is already shown in Section 3.2.9 except that the multiplication of the coefficients K1 and K2 is replaced with a shift right operation due to

the selection of K1 and K2 as the 2^N. The time tracker output LEAD_LAG goes to the clock generator. The I/O signals of the time tracker is shown in Table 4.10.

Table 4.10: I/O signals in time tracker

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Frame_edge_code	I	1	Frame edge for the code generators
Start_time_trk	I	1	Enable signal for the time tracker
Clk_sym_pl	I	1	Enable signal for the load of every CPICH symbol
Chipx_pl	I	1	Clock signal for the load of the moving-averaged CPICH symbol (15 KHz)
Chipx1	I	1	Chip clock (3.84 MHz)
Chipx8	I	1	1/8 chip clock (30.72 MHz)
Time_req	I	1	Requests the fine time tracking result at every frame
Time_trk_th	I	18	Threshold value for the comparison
P_demod_i_early	I	9	Demodulated I channel CPICH symbol in early position
P_demod_q_early	I	9	Demodulated Q channel CPICH symbol in early position
P_demod_i_late	I	9	Demodulated I channel CPICH symbol in late position
P_demod_q_late	I	9	Demodulated Q channel CPICH symbol in late position
Lead_lag	O	2	Time tracking result for clock generator

4.3 Finger on decision

A finger-on-decision block determines each finger's on/off condition based on the measured power level. Table 4.11 shows the input and output of the finger on decision block. We use two threshold values, OFF_TH and ON_TH, to determine if the finger should be turned on or off, and the threshold values are set by the upper layer. When MAG is smaller than OFF_TH, the decision flag FINGER_ON remains off until MAG has a value larger than ON_TH. Once the finger is turned on, the finger remains on until the measured power becomes lower than OFF_TH. The two different threshold values for fingers' on and off decisions prevent frequent changes, which works as a lowpass filter. Figure 4.32 shows an example situation with varying input power levels.

Table 4.11: I/O signals of finger decision block

Signal name	I/O	Bit	Description
Reset_sys	I	1	Reset
Frame_edge_code	I	1	Frame edge for the code generators
Chipx_pl	I	1	Clock signal for the load of the moving-averaged CPICH symbol (15 KHz)
Chipx1	I	1	Chip clock (3.84 MHz)
Chipx8	I	1	1/8 chip clock (30.72 MHz)
Mag	I	10	Estimated power level of each finger
Off_th	I	10	Finger off threshold
On_th	I	10	Finger on threshold
Finger_on	O	1	Finger on/off flag

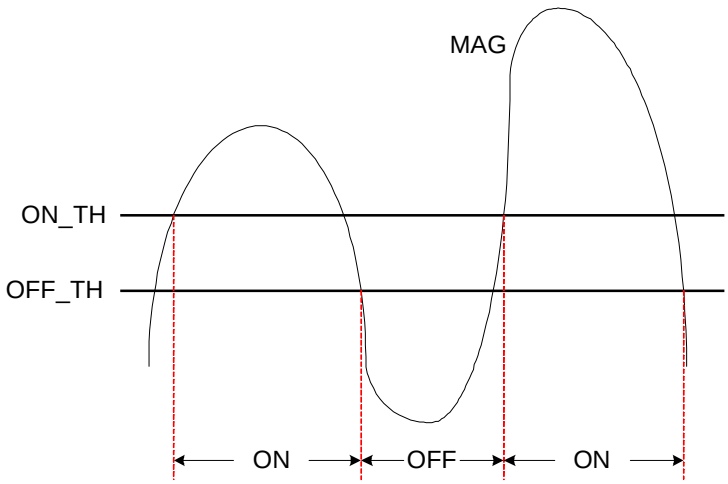


Figure 4.31: Finger on/off according to threshold levels

4.4 Deskew-combiner

The deskew-combiner is a combination of deskewer, combiner and SRAM controller. Thus, the deskew-combiner combines each finger's output, for which the delay is adjusted, and sends the combined data symbols to the SRAM.

The data load block, which is a sub block of the deskew-combiner, loads the compensated DPCH symbols from a dedicated finger. The number of data load blocks is equal to the number of fingers, and there are four data load blocks for our design.

The deskew-combiner has two register blocks, which are named EVEN and ODD, to support accidental conditions to be explained in Section 4.4.4. One of the loaded data

is selected by a priority encoder and sent to a register block. There are three modes for the dump of the data symbol that are stored in the register blocks to the SRAM, and they are NORMAL, NORMAL_FAST and FAST. The structure of the deskew-combiner is given in Figure 4.32, and each of the operations is discussed in detail in the following sub sections.

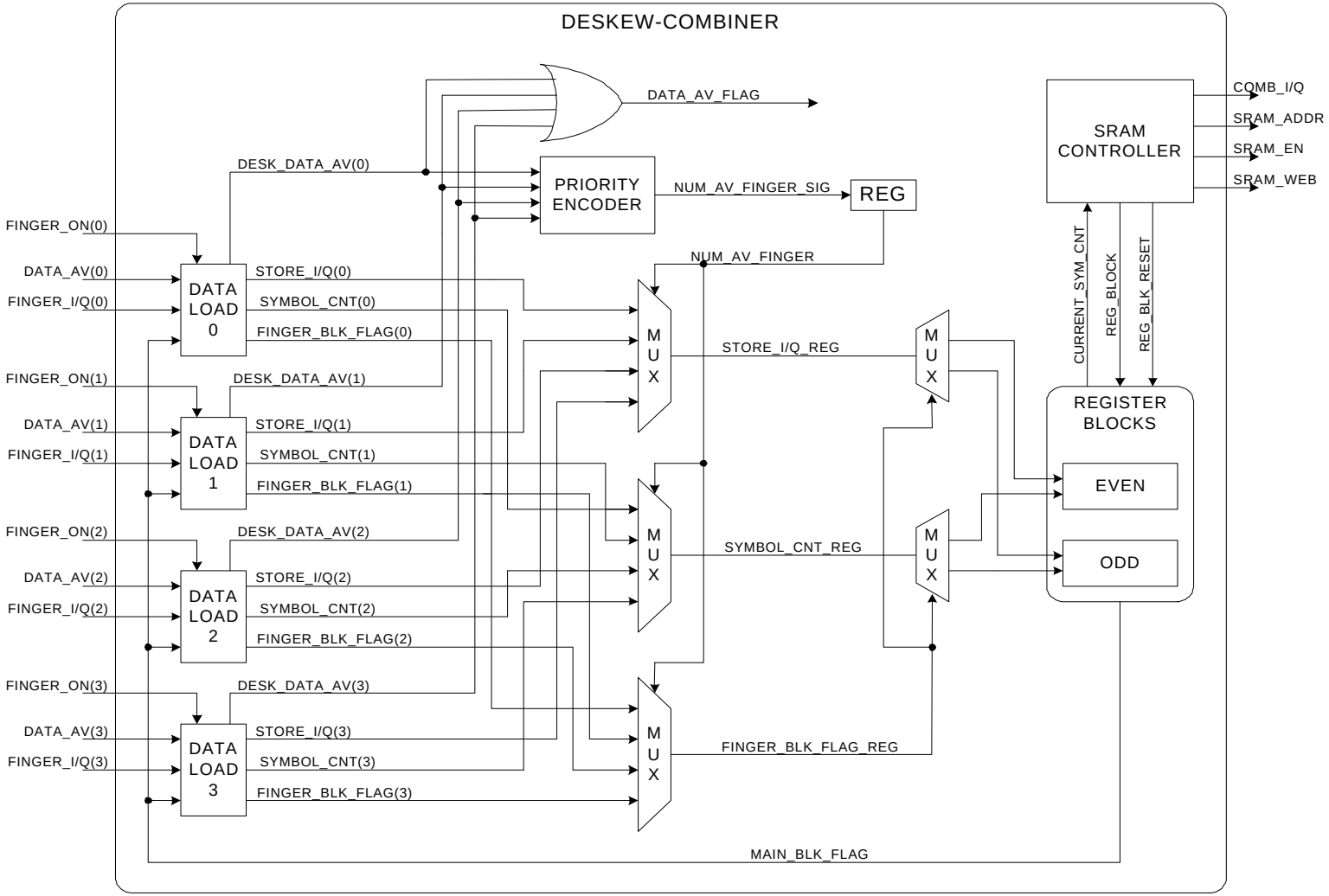


Figure 4.32: Deskew-combiner structure

4.4.1 Data load

The data load block stores the data symbol from the dedicated finger when the symbol is available. A finger indicates symbol availability of the data by setting DATA_AV to '1'. When the data symbol is available from a finger, data load block stores the data and notifies the deskew-combiner to combine it by setting DESK_DATA_AV to '1'. DESK_DATA_AV changes to '0' as soon as the deskew-combiner finishes handling the stored data by receiving COMB_DONE from the deskew-combiner. Table 4.12 and Figure 4.33 show the I/O signals in the data load block.

Table 4.12: I/O signals in the data load block

Signal name	I/O	Bit	Description
Reset_sys	I	1	Resets when the finger is unlocked or the receiver is turned off
Chipx8	I	1	1/8 chip clock (30.72 MHz)
Data_av	I	1	Indicates that a new data symbol from a dedicated finger is available
Comb_done	I	1	Indicates that the deskew-combiner finished combining the stored data in the data load block
Finger_on	I	1	Indicates that if the finger is locked or unlocked
Main_blk_flag	I	1	Indicates that the next prepared memory block is to be combined
Sf_ref	I	8	Spreading factor for current symbol
Finger_I	I	15	I channel output data from the dedicated finger
Finger_q	I	15	Q channel output data from the dedicated finger
Store_I	O	15	I channel stored data to be combined in the combine block
Store_q	O	15	Q channel stored data to be combined in the combine block
Symbol_cnt	O	14	The index of the symbol that is currently dealt with
Desk_data_av	O	1	Indicates that storing the data is finished, goes to the deskew control block
Finger_blk_flag	O	1	Indicator of the target register block for current symbol

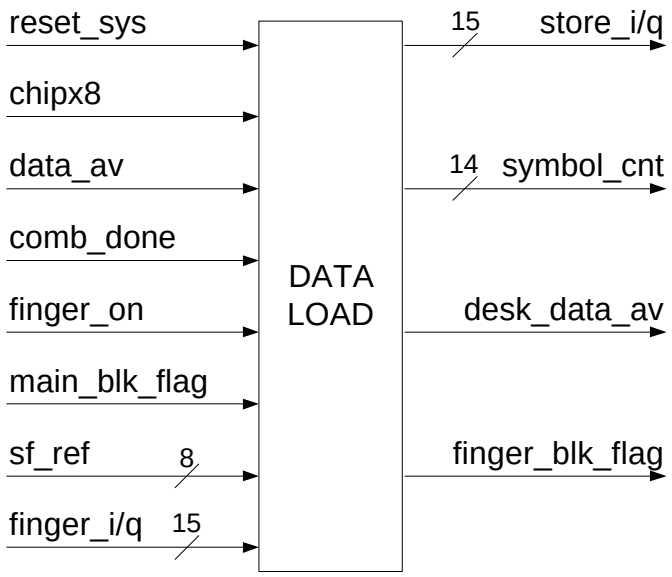


Figure 4.33: Data load block

4.4.1.1 Data load state machine

A finite state machine, which controls the operation of the data load block, is a *data load state machine*. The *data load state machine* is depicted in Figure 4.34. In the INITIAL state, if DATA_AV is ‘1’, which means that the data symbol from the finger is ready to be read, the data load state is changed to NOTICE_DESKEWER. In the NOTICE_DESKEWER state, the data symbol and spreading factor are loaded and the symbol counter starts counting the number of incoming symbols. When the received spreading factor is different from the previous one, or when the loaded data symbol is the first symbol of a new frame, the target register block index FINGER_BLK_FLAG loads the value of MAIN_BLK_FLAG to change the target register block. DESK_DATA_AV_SIG is set to ‘1’. Thus, if COMB_DONE is ‘1’, DESK_DATA_AV is set to ‘1’ to report the availability of the loaded data to the deskew-combiner.

Then, the state changes to WAIT_DONE. In the WAIT_DONE state, if COMB_DONE is ‘1’ the state changes to DONE and DESK_DATA_AV_SIG is set back to ‘0’ to prevent the unexpected activation of DESK_DATA_AV. In the DONE state, the state is set back to INITIAL when the DATA_AV signal goes to low in time to prevent

loading some symbols more than once. With this additional state, a finger does not need to care about the signal duration of DATA_AV.

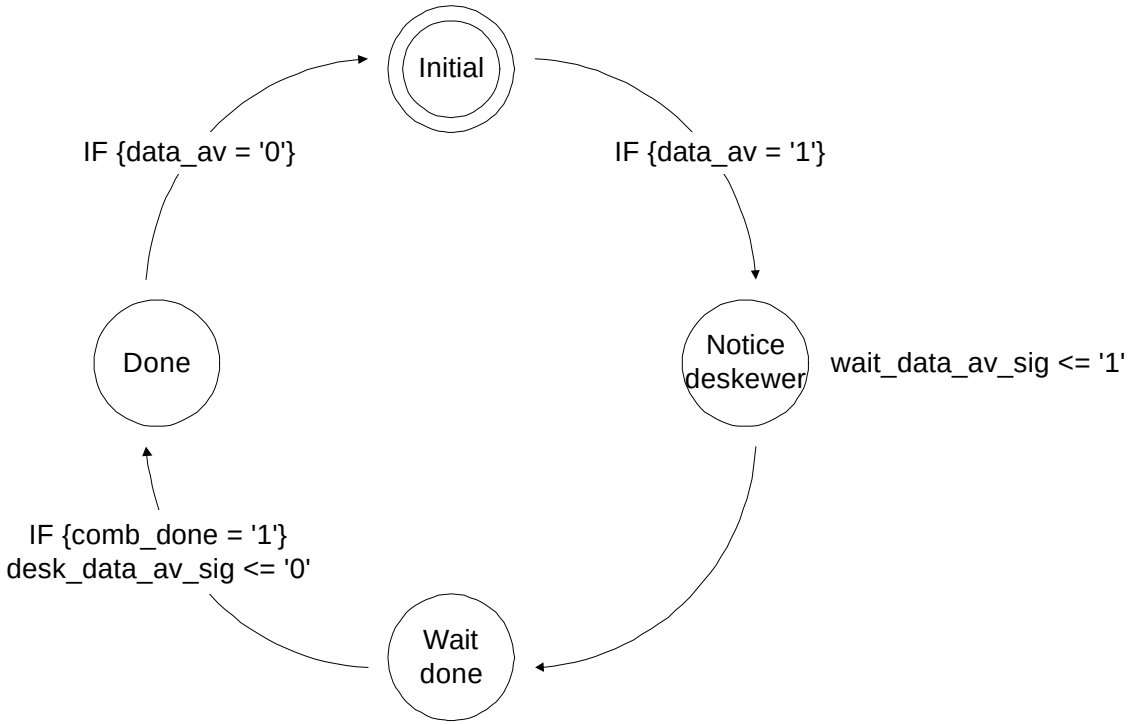


Figure 4.34: Data load state machine

4.4.2 Priority encoder

The deskew-combiner compiles the loaded data into a register block for the combining operation. The combining operation is applied to one of the loaded data, which is selected by the priority encoder. The data load block reports the availability of the data from a finger, but the data availability timing is not fixed and the order of the data availability is also variable because of the variable path delays. Thus the deskew-combiner should check which data load block has the available data symbol, and we use a priority encoder to avoid the scanning process for the whole data load blocks' outputs by checking DESK_DATA_AV of the 0th data load block to the higher ordered data load block. This prioritizing becomes more important as the number of the finger increases.

4.4.3 Deskew state machine

A *Deskew state machine* controls the deskew and combining operation. Figure 4.35 shows the *deskew state machine*. It starts from the INITIAL state. When at least one finger is ready for combining, the availability flag DATA_AV_FLAG is set to '1', thus the flag has the ORed value of all data load blocks' DESK_DATA_AV's. If the DATA_AV_FLAG is '1', the state is changed to START_COMB. At the same time, the register that indicates the available finger's number, NUM_AV_FINGER, loads the finger number, which is the output of the priority encoder. Also, COMB_DONE for the selected finger is set to '1' to inform that the combination operation for the loaded data is done. In the START_COMB state, if DATA_AV_FLAG is still '1', which means that another symbol from the finger is available, NUM_AV_FINGER is updated and COMB_DONE of the selected finger is set to '1' and those of other fingers are set back to '0'. Then, if there are no more available data from the fingers, the state goes back to INITIAL.

Whenever NUM_AV_FINGER is updated, SYMBOL_CNT, STORE_I/Q and FINGER_BLK_FLAG of the proper finger are also loaded to SYMBOL_CNT_REG, STORE_I/Q_REG and FINGER_BLK_FLAG_REG, respectively. These loading operations are performed using three MUX with NUM_AV_FINGER as a select input, as shown in Figure 4.32.

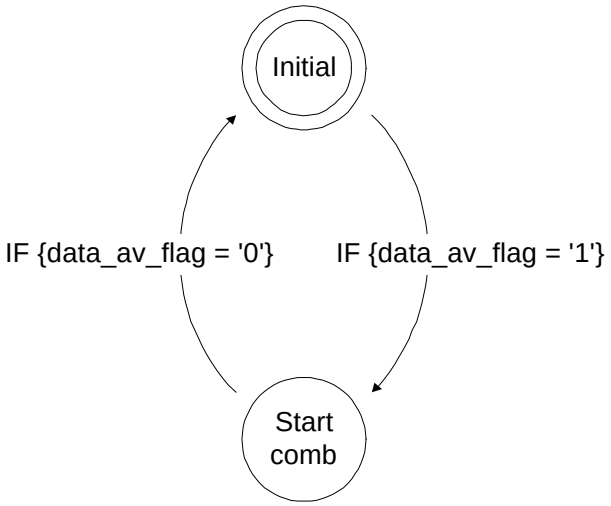


Figure 4.35: Deskew state machine

4.4.4 Register blocks

The loaded symbols, STORE_I_REG and STORE_Q_REG, of every finger are stored and added to the register block after adjusting the delay. The deskew-combiner has two register blocks and each of them has a length of eight.

First, the length of the register block is decided by the maximum difference of the arrival times of the symbols in fingers. The arrival time difference includes the difference between the multipath delays and the difference of the time offsets between the base stations. In this implementation, we assume that the maximum difference between the fingers' arrival time is 32 chips. The maximum 32-chip delay is equivalent to the maximum delay of eight symbols, because the spreading factor 4 provides the maximum symbol rate and when the spreading factor is four, 32 chips correspond to eight symbols. Therefore, the length of the register block is selected as eight. The length of the register block for the practical systems can be adjusted based on the results of the field measurements. From now on, we explain the deskew-combiner based on the assumption that the length of the register block is eight.

After determining the length of the register, we consider the unexpected arrival of frames. Unexpected arrival of the frame is shown in Figure 4.36. Figure 4.36 assumes that finger #3 is deactivated during frame #0 and newly activated at the start of the frame #1, and processes a path that arrives faster than other paths that are processed by the pre-locked fingers. This case can happen when the cell searcher locks a new multipath, which has shorter delay than other multipaths processed by the pre-locked fingers, for a previously deactivated finger. Also, after the handover, if the new base station assigns a smaller time offset value than the old base station, the arrival of the new frame is possibly faster than that of the previous frame.

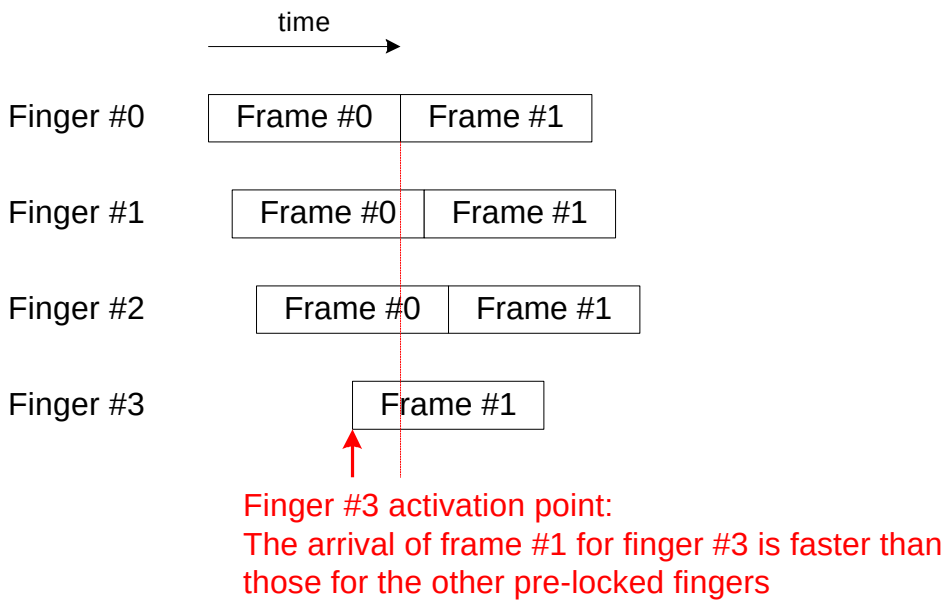


Figure 4.36: The unexpected arrival of the frames

To manage this case, we use two separate register blocks. One register block is used for frame #0 as shown in Figure 4.36, and the other register block is used for frame #1. If a single register block is used, the register block size should be sufficiently large not to lose any data. However, by using two separate register blocks for different frames, each register block size can be maintained as small as eight. The detailed structure for the register blocks is depicted in Figure 4.37.

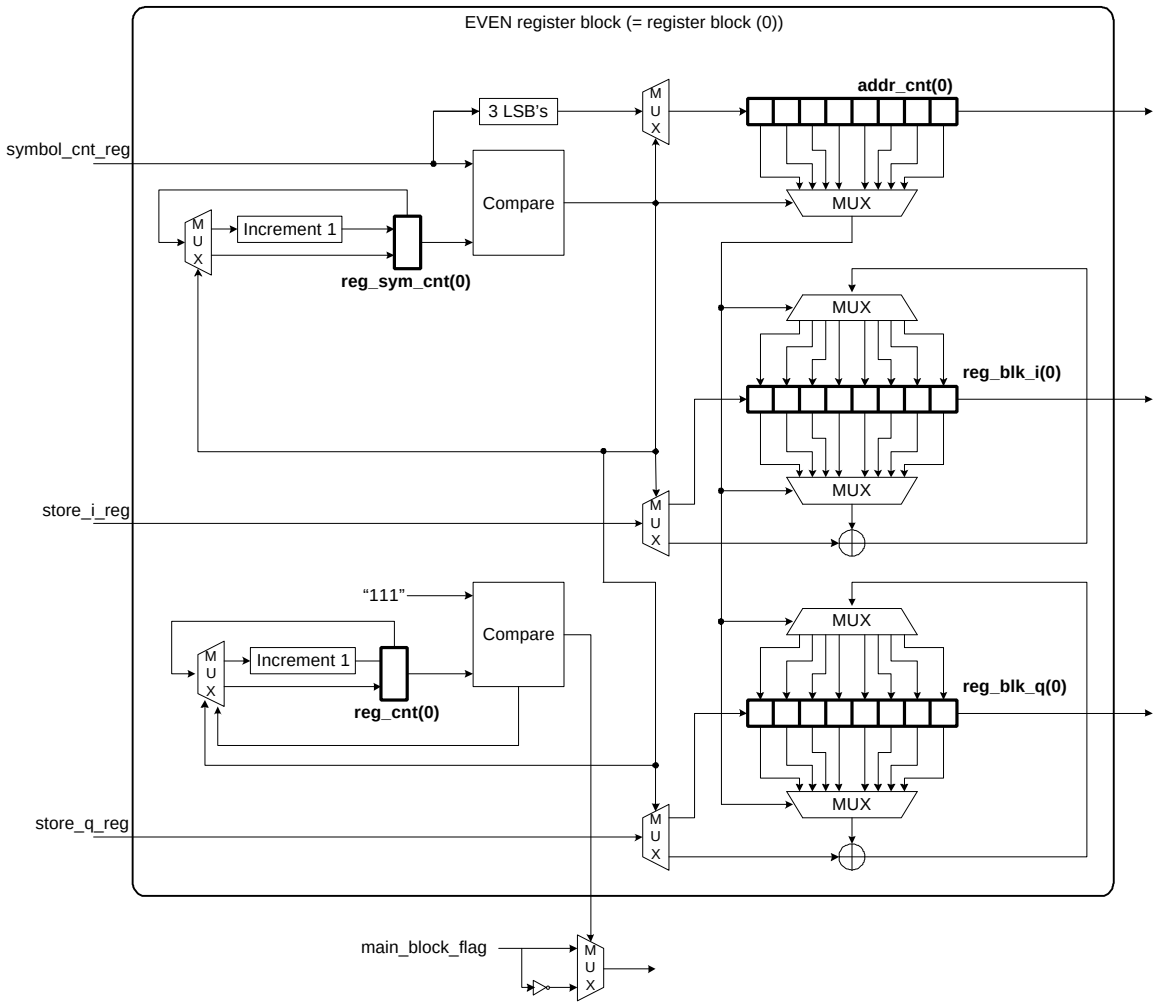


Figure 4.37: Register block

Each register block consists of REG_BLK_I, REG_BLK_Q, ADDR_CNT, REG_SYM_CNT and REG_CNT. REG_BLK_I and REG_BLK_Q are the registers storing the symbols from the data load block, and ADDR_CNT stores the address of REG_BLK_I/Q for the incoming symbols. REG_SYM_CNT has the expected new symbol number. Since SYMBOL_CNT_REG has the initial value 0, the initial value of REG_SYM_CNT is set to 1 to indicate the expected symbol number and avoid the any unnecessary operations. Thus, for example, if currently the register block has received two symbols from finger #0, three symbols from finger #1, 1 symbol from finger #2 and 3 symbols from finger #3, the value of REG_SYM_CNT is 4. Figure 4.38 shows this example graphically:

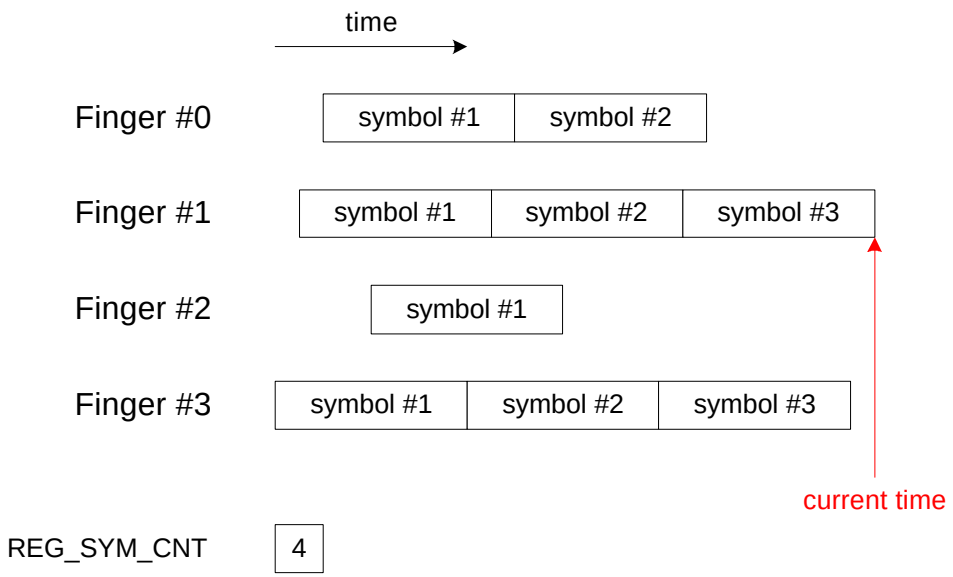


Figure 4.38: The value of REG_SYM_CNT according to the received symbols until current time

When the register block receives a symbol from a data load block, the value of REG_SYM_CNT is compared with the SYMBOL_CNT_REG of the received symbol. Then if SYMBOL_CNT_REG has the same value as REG_SYM_CNT, REG_BLK_I/Q is shifted right one element and the received symbol is stored in REG_BLK_I/Q(0). ADDR_CNT is also shifted right one element and SYMBOL_CNT_REG's least significant three bits are stored in ADDR_CNT(0). REG_SYM_CNT is incremented by one. This process is given as an example in Figure 4.39. Let us assume that finger #0 has received three symbols, fingers #1 and #2 have received two symbols, and finger #3 has received one symbol. The newly received symbol is the 4th symbol of finger #0. The received symbol's number, SYMBOL_CNT_REG, is compared with the value of REG_SYM_CNT. Since they have the same value that is four, ADDR_CNT is shifted right one element and SYMBOL_CNT_REG is stored in ADDR_CNT(0). REG_BLK_I/Q is also shifted right one element and the received symbol is stored in REG_BLK_I/Q(0). Finally, REG_SYM_CNT is updated to 5.

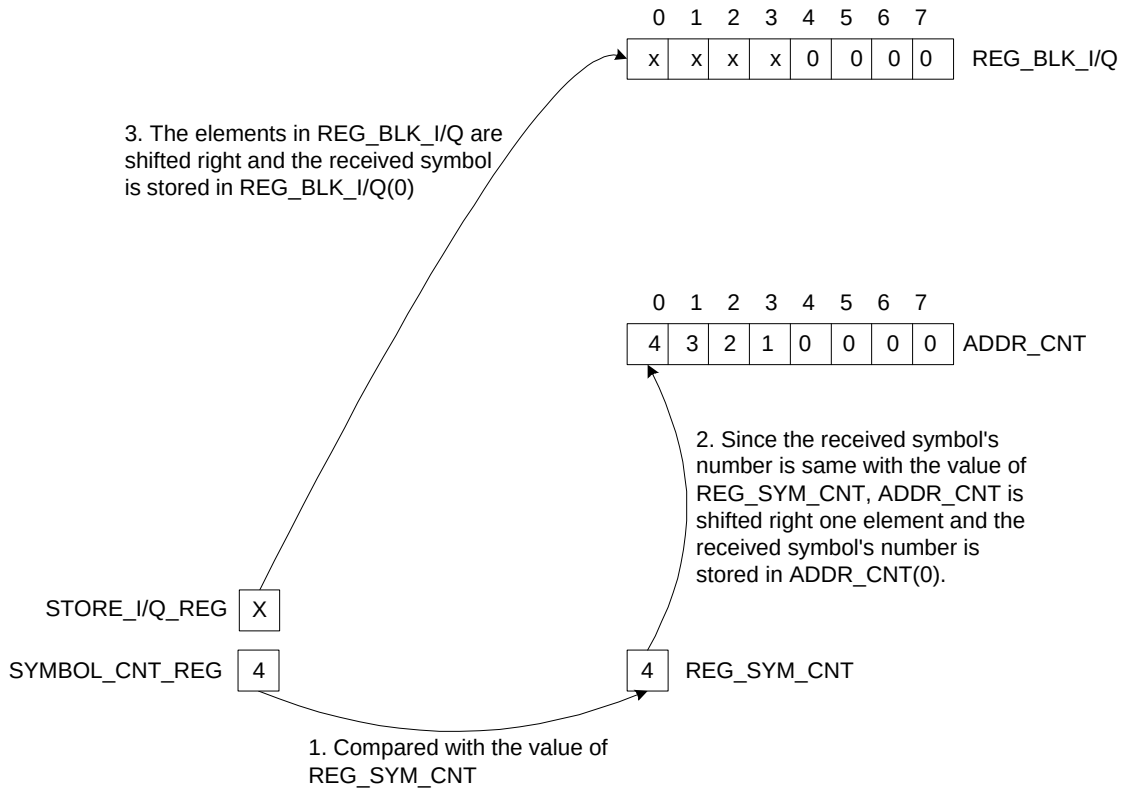


Figure 4.39: Operation of register block I

If SYMBOL_CNT_REG has a different value than REG_SYM_CNT, ADDR_CNT(SYM_CNT_REG(2 downto 0)) receives the address of REG_BLK_I/Q to which the received symbol should be combined. Then, the received symbol is added to the stored value in REG_BLK_I/Q(ADDR_CNT(SYM_CNT_REG(2 downto 0))). ADDR_CNT is not changed and REG_SYM_CNT maintains the current value. Figure 4.40 shows another example to depict this process. Let us assume that finger #0 has received four symbols, fingers #1 and #2 have received two symbols, and finger #3 has received 1 symbol. The newly received symbol is the 2nd symbol of finger #3. The received symbol's number, SYMBOL_CNT_REG, is compared with the value of REG_SYM_CNT. Since they are different, ADDR_CNT(SYMBOL_CNT_REG) is referred to get the target address of REG_BLK_I/Q, which is 2. Then, the received symbol is combined to REG_BLK_I/Q(2).

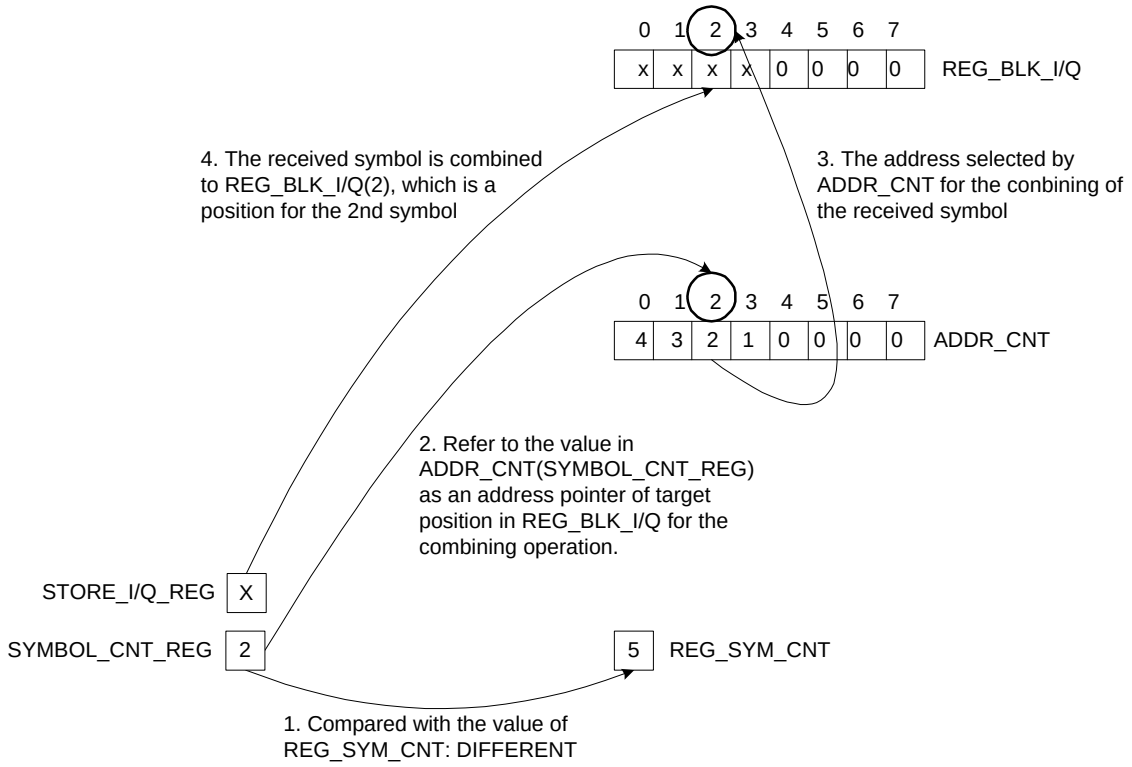


Figure 4.40: Operation of register block II

The example that shows the operation of the register block at the beginning of a frame is given in Table 4.13. We assume that the register block does not change during this example period and all of the symbol values are 1's. The process is explained below in time order:

- Initial state: REG_SYM_CNT is 1 and all others are 0.
- 1st received symbol: The received symbol is the 1st symbol from finger #0, which means that SYMBOL_CNT_REG is 1. The received symbol is stored in REG_BLK_I/Q(0), and ADDR_CNT(0) loads SYMBOL_CNT_REG(2 downto 0), because REG_SYM_CNT is the same as SYMBOL_CNT_REG. Then REG_SYM_CNT is incremented by 1, which results in 2. This means that the next expected symbol number is 2.
- 2nd and 3rd received symbols: The next two symbols are the 1st symbol of finger #1 and finger #2, respectively. Both of them are added to REG_BLK_I/Q(ADDR_CNT(SYMBOL_CNT_REG(2 downto 0))) because

the received symbol's SYMBOL_CNT_REG is not equal to REG_SYM_CNT. ADDR_CNT is not changed.

- 4th received symbol: The received symbol is the 2nd symbol from finger #0, and this value is the same as the value of REG_SYM_CNT. Thus, REG_BLK_I/Q is shifted right one element and the received symbol is stored in REG_GLK_I/Q(0). ADDR_CNT is also shifted right one element and ADDR_CNT(0) is changed to 2, which is the least significant three bits of SYMBOL_CNT_REG. Therefore, we can know that the 2nd symbol of the frame is compiled into the 0th element of the register block and the 1st symbol of the frame is stored in REG_BLK_I/Q(1). REG_SYM_CNT is incremented to 3, which means that the next symbol count of the frame is 3.
- 5th received symbol: The received symbol is 3rd symbol from finger #0. Since SYMBOL_CNT_REG is the same as REG_SYM_CNT, REG_BLK_I/Q is shifted right one element and the received symbol is stored in REG_BLK_I/Q(0). ADDR_CNT is shifted right one element also and ADDR_CNT(0) is changed to 3. Therefore, we can know from the address pointer that the 1st symbol of the frame is stored in REG_BLK_I/Q(2), the 2nd symbol of the frame is stored in REG_BLK_I/Q(1) and the 3rd symbol of the frame is stored in REG_BLK_I/Q(0).

The procedure will be continued as shown in Table 4.13.

Table 4.13: Process for the register block

Total number of the received symbols	SYMBOL_ CNT_REG	NUM_AV _FINGER	REG_BLK_I/Q								ADDR_CNT								REG_SYM _CNT	REG_ CNT
			0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7		
Initial state	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1 => 2	1
2	1	1	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	1
3	1	2	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	1
4	2	0	1	3	0	0	0	0	0	0	2	1	0	0	0	0	0	0	2 => 3	2
5	3	0	1	1	3	0	0	0	0	0	3	2	1	0	0	0	0	0	3 => 4	3
6	1	3	1	1	4	0	0	0	0	0	3	2	1	0	0	0	0	0	4	3
7	2	2	1	2	4	0	0	0	0	0	3	2	1	0	0	0	0	0	4	3
8	2	1	1	3	4	0	0	0	0	0	3	2	1	0	0	0	0	0	4	3
9	4	0	1	1	3	4	0	0	0	0	4	3	2	1	0	0	0	0	4 => 5	4
10	2	3	1	1	4	4	0	0	0	0	4	3	2	1	0	0	0	0	5	4
11	3	2	1	2	4	4	0	0	0	0	4	3	2	1	0	0	0	0	5	4
12	5	0	1	1	2	4	4	0	0	0	5	4	3	2	1	0	0	0	5 => 6	5
13	3	3	1	1	3	4	4	0	0	0	5	4	3	2	1	0	0	0	6	5

As shown in the above examples, ADDR_CNT and REG_SYM_CNT use only three LSB's of SYMBOL_CNT_REG. In our design, when a finger is turned off in the middle of the frame, that finger is not turned on until a new frame begins. Thus we can assume that the order of the incoming symbols of one frame to the deskew-combiner does not change. Therefore, the symbol number of the received symbol at the deskew-combiner increases monotonically, and this makes REG_SYM_CNT able to store the expected symbol number. We also assume that the maximum difference between the delays of the multipath is eight, thus the symbols that have same three LSB's cannot come in during the same eight-symbol period. Therefore, ADDR_CNT can use only three LSB's of SYMBOL_CNT_REG. As long as the length of the register block is chosen as 2^N, ADDR_CNT can take only N LSB's of SYMBOL_CNT_REG.

Now, let us consider changing the register block. Each of the register blocks is for one frame, thus at the beginning of the new frame the register block index should be flipped to indicate the other register block. A counter REG_CNT controls this process. REG_CNT counts the symbol number of a frame to have the largest value among the received SYMBOL_CNT_REG's until it has the largest value, which is seven in case the register length is eight. In other words, REG_CNT, which is a 3-bit register, indicates whether the registers for the I and Q symbols, REG_BLK_I/Q are full or not. When REG_CNT has the maximum value, it stops incrementing and flips the register block selection flag, MAIN_BLK_FLAG. MAIN_BLK_FLAG is loaded to the register block selection flag for each of the fingers, FINGER_BLK_FLAG, in the data load block when a new frame is starting, so that the symbols of different frames are not mixed together. The entire process of the register block is given as a flowchart in Figure 4.41.

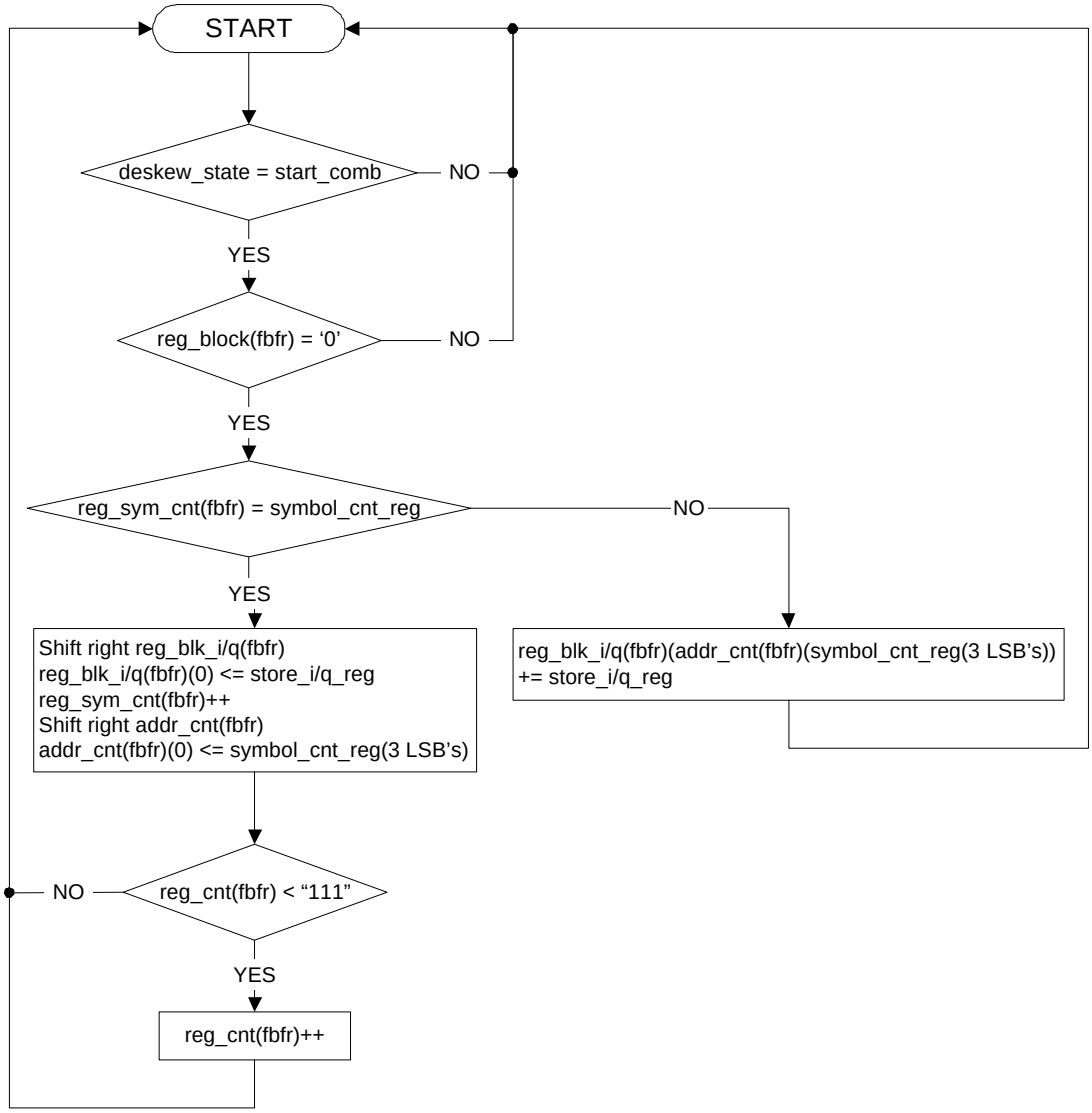


Figure 4.41: Flowchart for the process of the register block (fbfr: FINGER_BLK_FLAG_REG)

By using the address pointer, priority encoder and two register blocks, the deskew-combiner can dynamically support the variable arriving times of the multipaths within the range of the maximum difference of symbol delay. Conventionally, the number of necessary registers is equal to the number of fingers to store the symbols of each finger. Therefore, our design is more efficient than the conventional design for the rake receiver that has more than two fingers, because our design can support a large

number of fingers with a small number of registers, and the combining operation can be done with the deskew operation at the same time.

4.4.5 SRAM controller

The stored and combined symbols in the register blocks should be dumped to the SRAM, which is a gateway between the upper layer and the rake receiver. We designed the SRAM to have two memory banks, each of which is for one frame. Since the maximum symbol number of one frame is 9600, when the spreading factor is 4, the SRAM address is 15 bits. The MSB is for selection of the memory bank and the other 14 bits represent the address for each of the memory banks.

4.4.5.1 Dump mode

The deskew-combiner provides three dump modes depending on the register block's status. Figure 4.42 shows the *dump mode state machine*. DUMP_POSITION represents the index of the REG_BLK_I/Q of the selected register block for dump position. It is initially set to the largest value to indicate the last element of the register, which is seven in case the length of the registers for I and Q symbols is eight. DUMP_BLK represents the index of a register block, whose symbols in REG_BLK_I/Q are currently dumped to the SRAM. REG_BLOCK is a flag to prevent the selected register block from being written during the fast dump mode.

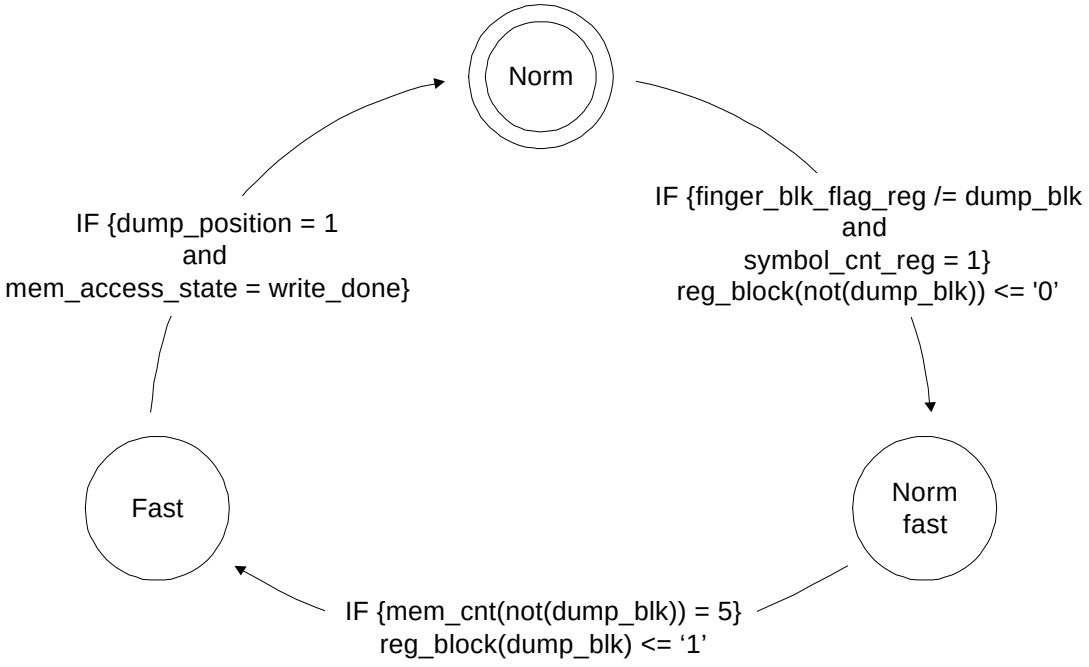


Figure 4.42: Dump mode state machine

Initially, the dump mode is NORMAL. Let us assume that EVEN register block is in use. In NORMAL state, the last elements of REG_BLK_I/Q of the EVEN register block are dumped to the SRAM, because the dump position is assigned as seven. If FINGER_BLK_FLAG_REG is not the same as DUMP_BLK and SYMBOL_CNT_REG is 1, which means that the corresponding finger processed the first symbol of a new frame, the dump mode is changed to NORMARL_FAST and the blocking flag REG_BLOCK of the ODD register block is set to 0 to enable the ODD register block to receive the inputs.

In NORMAL_FAST mode, both of the register blocks are used to deal with the data from different fingers and different frames. NORMAL_FAST mode is needed to manage the newly activated fingers and handover. When a finger state is changed from off to on at the beginning of the new frame and it receives a multipath that has a smaller delay than the smallest delay of the previous frame's multipath delay, the EVEN register block is used for the previous frame and the ODD register block will be used for the new frame data at the same time. Also, after the handover, if the new base station assigns a smaller time offset for the mobile than the time offset of the previous base station, both of the register blocks will be used to support the process of two different frame's data. Thus,

the EVEN register block deals with the data of the old frame and the ODD register block handles the data of the new frame in the NORMAL_FAST mode. The EVEN register block dumps the elements in the last position, because DUMP_POSITION is still 7. However, when the ODD register block contains six symbols from the new frame, which means that the ODD register block should dump the stored data to the SRAM in at least two-symbol periods, the dump mode should be changed to FAST mode. As mentioned at the beginning of section 4.4, SRAM has two memory banks and only one of them can be accessed by the deskew-combiner at a time. Thus, 2 symbols before the dump action of the ODD register block, the dump mode is changed to FAST and REG_BLOCK of the EVEN register block is set to 1 to prevent receiving inputs.

In FAST mode, DUMP_POSITION is decremented by one at every memory access cycle, which is four CHIPX8 cycles, to dump out the stored data in the dump position that is indicated by DUMP_POSITION rather than waiting for the shifted data at the end of the register. Since the length of the register block is eight and there remain seven symbols to be dumped at the start of the FAST mode, it takes only 28 (= 4*7) CHIPX8 cycles to dump out all of the stored data in the EVEN register block. As stated, the dump mode transition timing is selected as a two-symbol period, which is eight chips or 64 CHIPX8 clock cycles with the spreading factor of 4. A 64 CHIPX8 cycle is much longer than the actual clock cycle taken for evacuating the EVEN register block. Thus the dump mode transition timing can be selected as only 1 symbol period, but we gave enough of a safety margin to make the operation sure. When a spreading factor larger than 4 is used, the safety margin is even larger. When the EVEN register block is evacuated and the *memory access state machine* is in the WRITE_DONE state, the dump mode goes to NORMAL for the ODD register block. *Memory access state machine* is covered in section 4.4.5.2.

4.4.5.2 SRAM access state machine

The *SRAM access state machine*, which is shown in Figure 4.43, handles the operation of the SRAM controller. In INITIAL state, if REG_DUMP_CNT and CURRENT_SYM_CNT have the same value or the dump mode is FAST, the state

changes to SEND_DATA. Let us assume that the currently used register block is EVEN. As described in section 4.4.5.1, when the dump mode is not FAST, the data stored in REG_BLK_I/Q(0)(7) is dumped to the SRAM. Since our deskew-combiner can receive the symbols in variable time order, the register block does not shift the elements in REG_BLK_I/Q to right whenever it receives the new symbols. Thus, we use two counters, REG_DUMP_CNT and CURRENT_SYM_CNT, not to dump the same symbols multiple times. REG_DUMP_CNT and CURRENT_SYM_CNT are 3-bit counters. REG_DUMP_CNT has an initial value 1 and CURRENT_SYM_CNT has the value of the last element of the currently used register block's ADDR_CNT, which is ADDR_CNT(0)(7). CURRENT_SYM_CNT remains 0 until REG_BLK_I/Q(0)(7) has a symbol value and equivalently ADDR_CNT(0)(7) has the value 1. At this moment, since we are assuming that the maximum delay difference between the fingers is 32 chips, which is eight symbols with spreading factor of 4, we can easily figure out that the combining operation for the symbols in REG_BLK_I/Q(0)(7) is already finished. Now, REG_DUMP_CNT and CURRENT_SYM_CNT have the same value, thus the state moves to SEND_DATA. If the dump mode is FAST, whether the values of REG_DUMP_CNT and CURRENT_SYM_CNT are same or not, the stored symbol is dumped according to DUMP_POSITION at every access cycle, decrementing the DUMP_POSITION by 1. Since the register block is prevented from being written, no more shift is guaranteed. Then, the state also changes to SEND_DATA.

In SEND_DATA state, the SRAM write enable signal is set to 0 to activate the writing operation. Then it changes to WRITE_DONE state to inform that the writing operation is finished. The SRAM write enable signal SRAM_WEN is set back to 1 to disable the write operation and REG_DUMP_CNT is incremented by 1. In WRITE_DONE state, we have two cases. One is that DUMP_POSITION is 1, which means that all of the stored symbols in REG_BLK_I/Q are written to the SRAM and one frame is finished, and the other is when DUMP_POSITION is not 1, which means that the current frame is still in process. In first case, a DUMP_BLK that indicates the current register block whose stored symbols are dumped to the SRAM is changed to point out the other register block, DUMP_POSITION is set to 7 to be prepared for the NORMAL dump mode and register block reset signal is generated to reset the current register block.

The state is changed to CHANGE_SRAM_ADDR. The second case decrements DUMP_POSITION by 1 when the dump mode is FAST, and changes the state to WRITE_COMPLETE. Then, the state is changed to INITIAL. In the CHANGE_SRAM_ADDR state, the MSB of SRAM_ADDR_REG is flipped to indicate the other memory bank of the SRAM and the other 14 bits are set to 0 to start the new address. In the WRITE_COMPLETE state, SRAM_ADDR_REG is just incremented by 1. The MSB of SRAM_ADDR_REG is the memory bank enable signal SRAM_EN and the other 14 bits of SRAM_ADDR_REG are the address of the selected memory bank, which is SRAM_ADDR. Then, the state goes back to INITIAL.

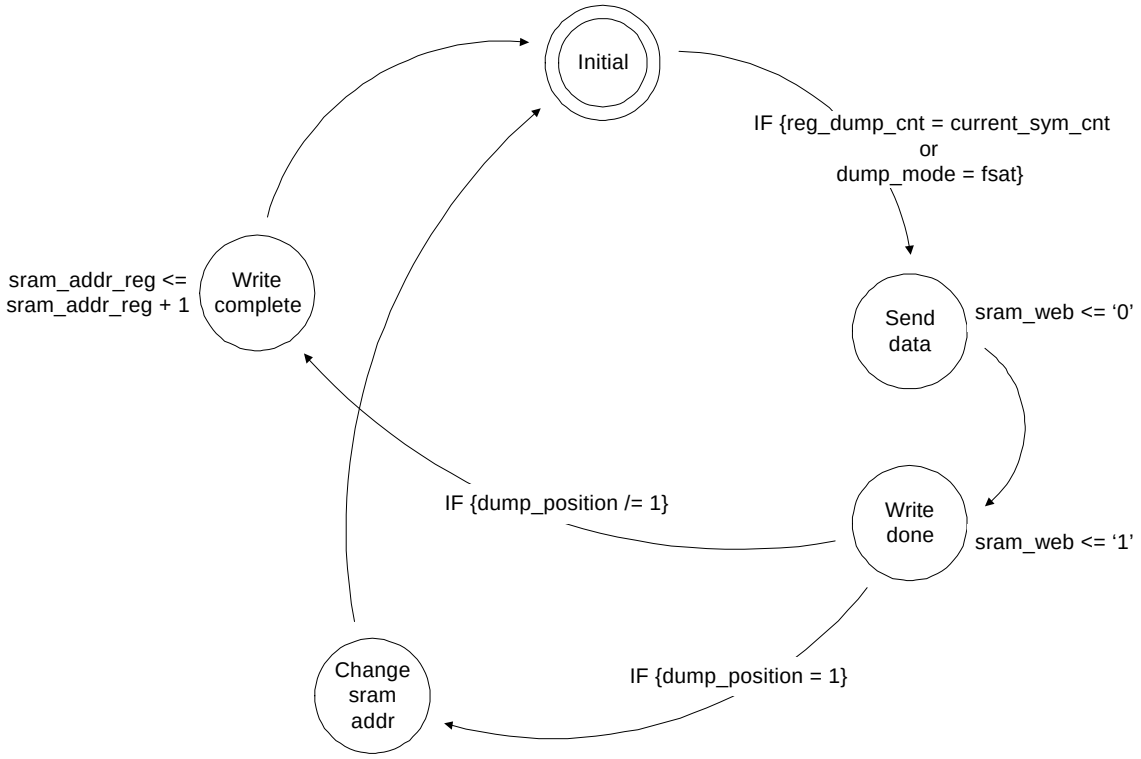


Figure 4.43: SRAM access state machine

Chapter 5 Experimental Results

In this chapter, we present the experimental results of the RT-level design of our rake receiver. We obtained the synthesized RT-level circuit from the VHDL design described in Chapter 4. The experimental results including the power consumption and the circuit complexity were measured from the synthesized circuit.

5.1 Environment

For the measurement of the power dissipation and the circuit complexity, we used Cadence® Ambit® BuildGates®. The cell library used in our experiments was TSMC 0.18 μm technology with 1.8V supply voltage obtained from Artisan. Ambit BuildGates provides several Low Power Synthesis (LPS) options, and it provides both power estimation and optimization capabilities under user supplied timing and area constraints. The LPS design flow is shown in Figure 5.1.

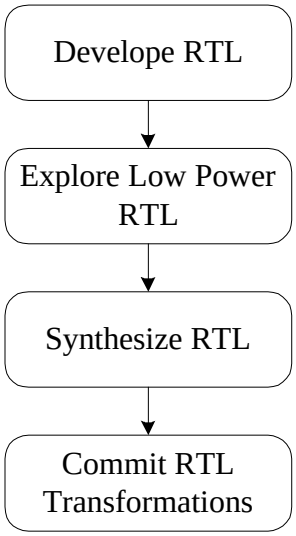


Figure 5.1: The Cadence LPS design flow [Cad01]

Using Cadence LPS option, designers need to develop an RTL code without being conerned about power. In the second step, LPS explores thousands of RTL

transformations for reducing power in the design. Then, after synthesizing the TRL code, LPS minimizes the power by committing the RTL transformations and performing gate-level power optimizations while satisfying the timing constraints [Cad01].

5.2 Description of test input sequences

Using Ambit BuildGates, we measured the power dissipation and the circuit complexity at the RT-level. For RTL simulation of the rake receiver, we generated the test input sequences from our MATLAB design. The received signals represented in the fixed-point format in MATLAB converted into 6-bit long binary numbers. These received signals in fixed-point numbers were saved in a file and applied to our rake receiver. The main clock, which is CHIPX8 of 30.72 MHz with 50% duty cycle, was applied to the rake receiver, from an external clock source. Memory blocks are excluded from the power and area estimation.

5.3 Power estimation methods

Power consumption for a circuit is composed of both static and dynamic power [Roy00]. Static power is generally dependent on the logical state of the cell. It is the power dissipated while signals are not switching. Static power includes leakage power and standby power. In CMOS logic, the leakage power is caused by sub-threshold leakage current, and the standby power by the DC current continuously drawn from power to ground. The standby power dissipation is insignificant for CMOS digital circuits.

Overall dynamic power of a cell is dependent on the supply voltage, temperature, load, and input slew rate, as well as the cell's state [Cad01]. Dynamic power dissipation is caused by switching activities of the circuits. It includes two types of power: short-circuit power and capacitor charging / discharging power. Short-circuit power is dissipated when a short that cannot be instantaneously turned on and off exists between the voltage supply and a ground rail created during output transitions. Capacitor charging / discharging power is caused by the charging and discharging of the entire capacitive

load and is the major source of power dissipation in CMOS circuits [Yea01]. A capacitive load includes internal capacitance, capacitance of the net, and the capacitance of the input / output pins.

We measured the power dissipation from the simulation of the test input sequences. Ambit BuildGates provides three types of power: leakage power, internal power, and capacitive load power. Leakage power, which is explained above, is a type of static power dissipation. Internal power consists of the short-circuit power of internal cells. Capacitive load power (net power) is calculated from the formula

$$P = aC_L V^2 f \tag{5.1}$$

where a is the switching activity, C_L is the capacitive load, V is the supply voltage, and f is the clock frequency. Ambit BuildGates computes the power at each node from the TSMC 0.18 μm library cell characteristics and the toggle information at each node of the gate-level model. Summing the power dissipations at each node results in the total power for the entire circuit.

5.4 Power dissipation

We first measured the power dissipation of the rake receiver using the method described in the previous sections. The experimental results are as follows: internal cell power is 29.1826 mW, leakage power is 7.342 μW, and net power is 27.0156 mW. The total power dissipation in the rake receiver is 56.2056 mW. Table 5.1 shows the power dissipation of the internal blocks of our rake receiver. MULTI_FINGER, which consists of four fingers, consumes 59 % and the deskew-combiner dissipates 33 % of the total power. The input buffer, DESK_RESET_GEN block, and other miscellaneous parts are responsible for consuming the remaining power. Figure 5.2 shows the power dissipation ratios of the internal blocks of the rake receiver.

Table 5.1: Power dissipation of the internal blocks of a rake receiver

Internal block name	Power dissipation (mW)	Percentage (%)
IN_BUFF	0.09	0.16
MULTI_FINGER	33.17	59.01
DESKEWER	18.79	33.44
DESK_RESET_GEN	0.01	0.02
Miscellaneous parts	4.24	7.54
Total (Rake receiver)	56.21	100.00

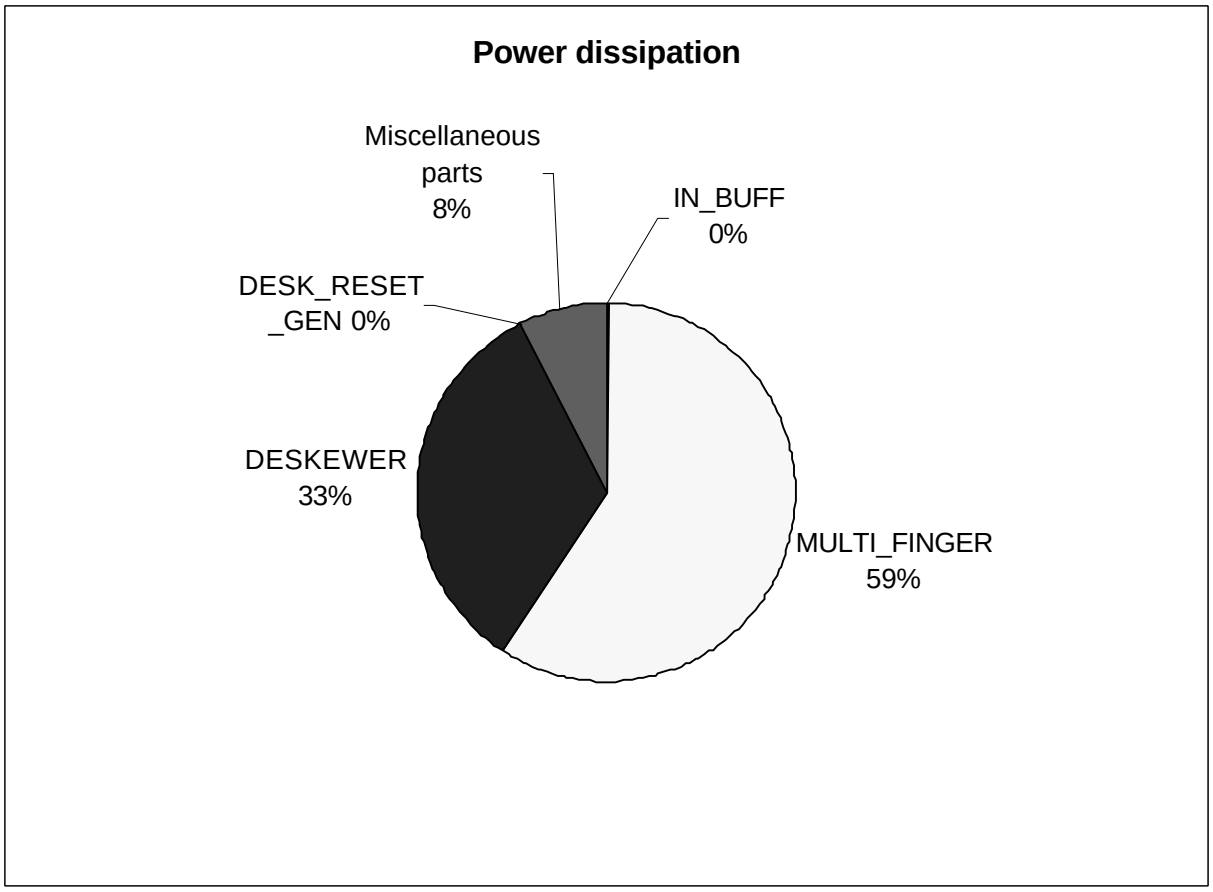


Figure 5.2: Power dissipation ratios of the internal blocks of a rake receiver

Table 5.2 shows the power dissipation of the individual blocks of a finger. The major power consuming blocks are the frequency offset estimator (FREQ_EST), the time tracker (TIME_TRK), and the clock generator (CLK_GEN). These three blocks are responsible for over 78 % of the total power dissipation. The large power consumption of these three blocks is due to the following reasons: the frequency offset estimator has

four multipliers and two accumulator, which cause the long outputs; The time tracker also has an accumulator that leads to a large internal signal; The clock generator sends output the generated clock signals to all the internal blocks of the finger, which causes the large fan-out and output load. Even though the lengths of the internal signals and the outputs are small, the large fan-out results in significant power consumption. Figure 5.3 shows graphically the power dissipation ratios of the internal blocks of a finger. The graph shows that the frequency offset estimator, the time tracker, and the clock generator are the most power-hungry blocks.

Table 5.2: Power dissipation of the internal blocks of a finger

Internal block name	Operating frequency (MHz)	Power dissipation (μW)	Percentage (%)
CLK_GEN	30.72	1553.3	18.77
CTRL_GEN	30.72	166.8	2.02
DEMOD (4)	3.84	231.3	2.80
AVG	3.84	68.2	0.82
CF_EST	0.015	10.4	0.13
COMP	3.84	67.2	0.81
FREQ_EST	30.72	3269.4	39.51
POW_EST	30.72	731.3	8.84
TIME_TRK	30.72	1712.3	20.69
FINGER_DECISION	30.72	88.7	1.07
OVSF_GEN (2)	3.84	91.5	1.11
SCRMAB_GEN (2)	3.84	89.1	1.08
Miscellaneous parts	N/A	196.1	2.37
Total (FINGER)	30.72	8275.6	100.00

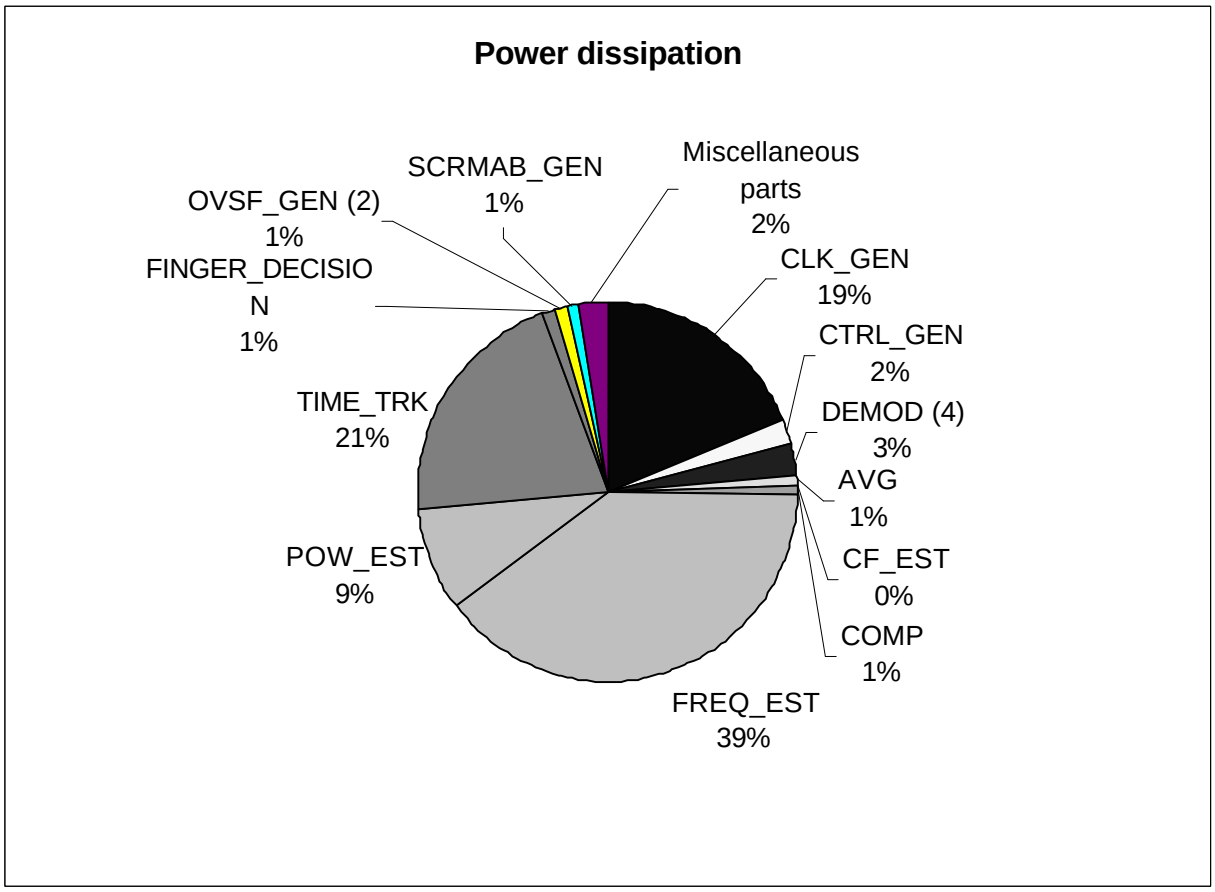


Figure 5.3: Power dissipation ratios of the internal blocks of a finger

5.5 Circuit complexity

The circuit complexity of our rake receiver was estimated in terms of the number of equivalent NAND2 gates. The total area of the rake receiver is about 1.0 million equivalent NAND2 gates. As can be seen in Table 5.3, one finger takes about 21.6 % of the total area and the deskew-combiner takes about 10 %. Figure 5.4 shows the area ratios of the internal blocks of the rake receiver.

Table 5.3: Area of the internal blocks of a rake receiver

Internal block name	Area (# of equivalent NAND2 gates)	Percentage (%)
IN_BUFF	300	0.03
MULTI_FINGER	849,656	86.39
DESKEWER	97,681	9.93
DESK_RESET_GEN	64	0.01
Miscellaneous parts	35,781	3.64
Total (Rake receiver)	983,482	100.00

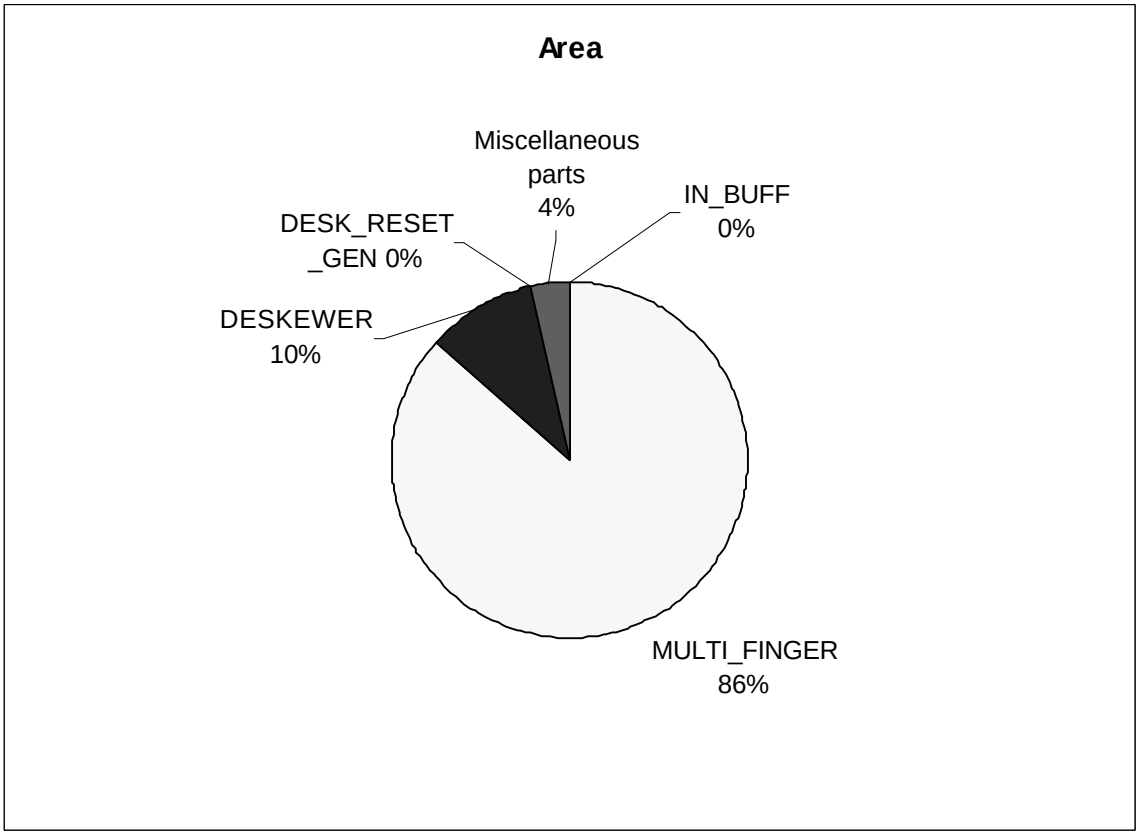


Figure 5.4: Area ratios of the internal blocks of a rake receiver

Table 5.4 provides the circuit area of the internal blocks of a finger. The frequency offset estimator (FREQ_EST) and the compensator (COMP) have the largest areas in the order. The two blocks are responsible for about 51 % of the total area. The frequency offset estimator contains four 9×9 multipliers, which cause high circuit complexity. The outputs of the multipliers are applied to accumulators and the outputs are not truncated, which results in 27 bits. The compensator has the same number of

9×9 multipliers, and their outputs are applied to the adders. However, the outputs of the adders are truncated to 15-bits, which results in less area than the frequency offset estimator. We replaced multipliers with a multiplexer and an adder by a shift operation for the power estimator, which reduced the area of the block to only 4 % of the total area. Without such replacements, we believe that the area of the power estimator would have been about 10% of the total area, as the original design needed two 9-bit multipliers. Figure 5.5 shows the circuit complexity ratios of the internal blocks of a finger in terms of the percentage.

Table 5.4: Area of the internal blocks of a finger

Internal block name	Area (# of equivalent NAND2 gates)	Percentage (%)
CLK_GEN	11,837	5.57
CTRL_GEN	2,922	1.37
DEMOD (4)	34,945	16.43
AVG	11,567	5.44
CF_EST	1,430	0.67
COMP	43,255	20.34
FREQ_EST	64,967	30.55
POW_EST	7,882	3.71
TIME_TRK	20,840	9.80
FINGER_DECISION	1,094	0.51
OVSF_GEN (2)	1,977	0.93
SCRMAB_GEN (2)	1,237	0.58
Miscellaneous parts	8,733	4.11
Total (FINGER)	212,686	100.00

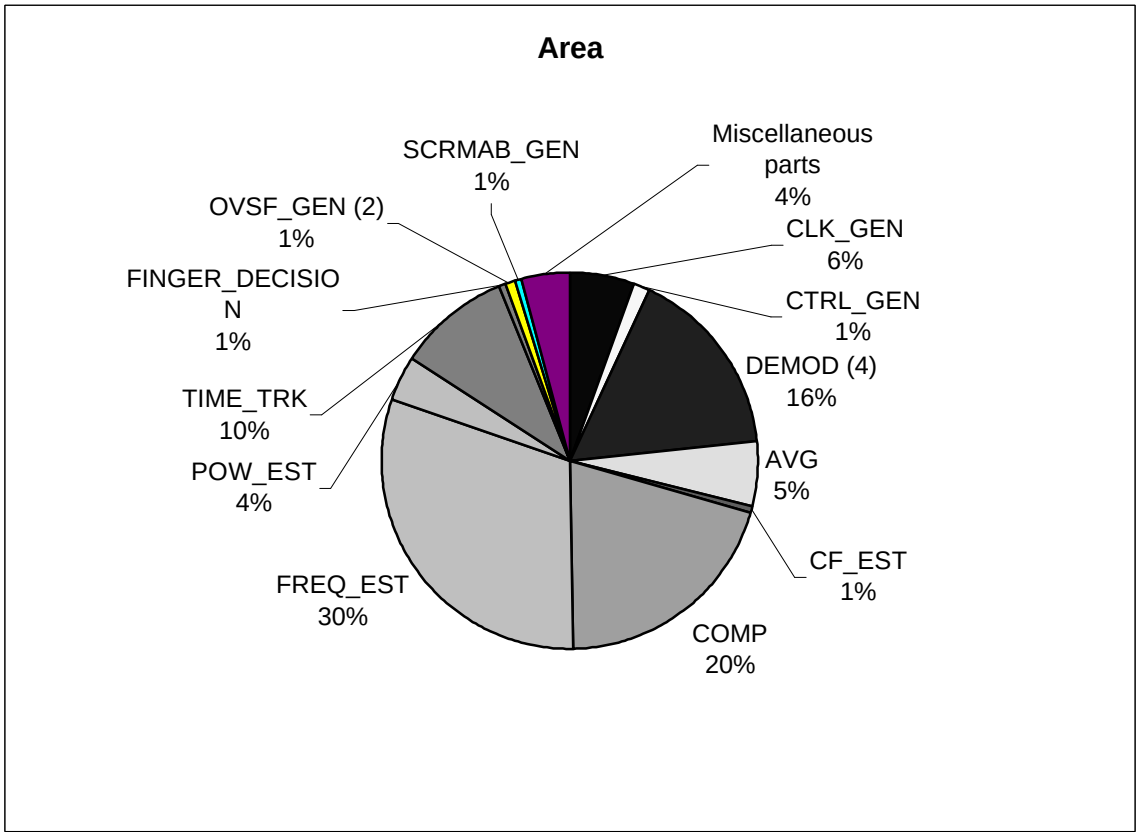


Figure 5.5: Circuit complexity ratio of the internal blocks of a finger

5.6 Power vs. Area

Figure 5.6 illustrates the relationship between the power dissipation and the circuit complexity for the internal blocks of a finger. As can be seen in the figure, the compensator and the frequency offset compensator cause substantial areas, but the power dissipation ratios are very different. As we discussed in Section 5.5, the frequency offset estimator has a larger area than the compensator. However, the compensator consumes much less power than the frequency offset estimator, because the compensator operates at a frequency eight times slower than that of the offset estimator. As described in (5.1), a higher operating frequency causes higher power consumption. This can be observed in Table 5.2. All blocks that operate with 3.84 MHz or 15 KHz, which are the slower operation frequencies, consume only about 1 % or even less than 1 % of the total power dissipation. Also, the clock generator dissipates about 19 % of the total power, while the

area taken by the clock generator is only about 6 %. As discussed in Section 5.4, the large power consumption is due to large fan-out, which results in large capacitive loads. This is a good example of the operating frequency constituting a major factor in the power consumption, and the capacitive load is also a considerable factor.

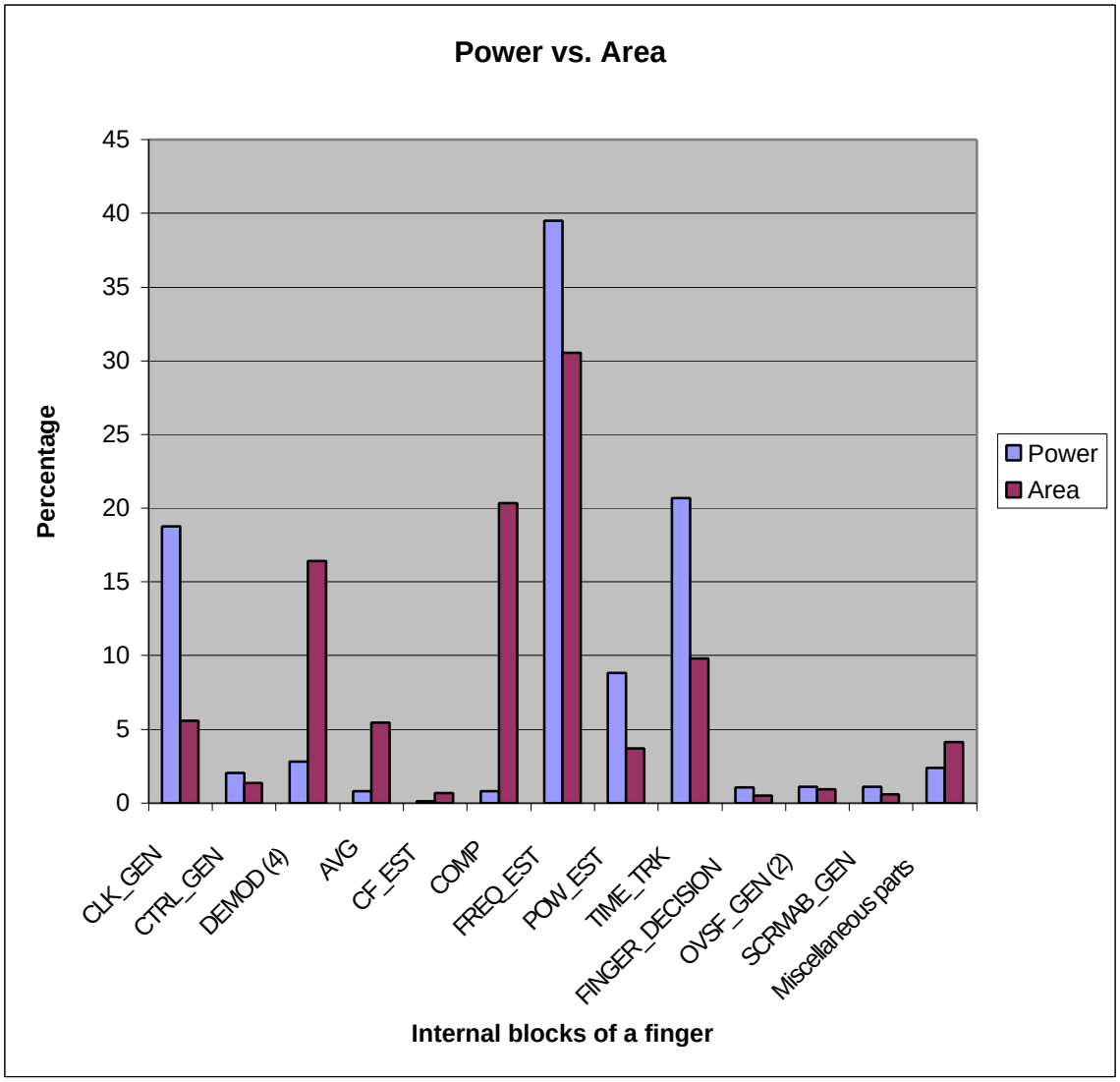


Figure 5.6: Power dissipation vs. circuit complexity of the internal blocks of a finger

Figure 5.7 depicts the relationship between the power dissipation and the circuit complexity for the internal blocks of the rake receiver. The ratio of power consumption

to area is less for fingers than for the deskew-combiner, as the average operating frequency of the fingers is four times slower than that of the deskew-combiner.

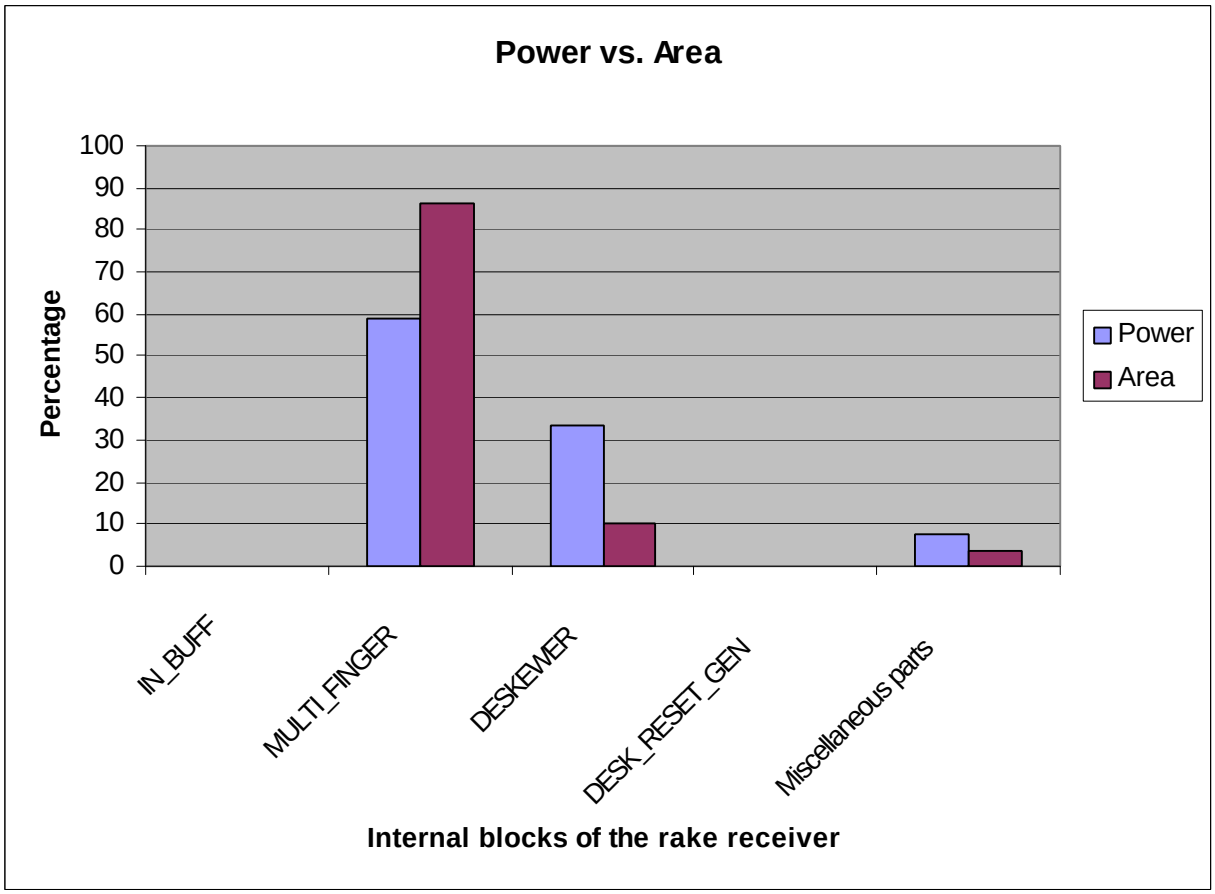


Figure 5.7: Power dissipation vs. circuit complexity of the internal blocks of the rake receiver

5.7 Concluding remarks

In this chapter, we examined the power dissipation and the circuit complexity. The cell library TSMC 0.18 μm technology with 1.8V supply voltage obtained from Artisan was used. In the next chapter, we conclude our work on the design of the WCDMA rake receiver.

Chapter 6 Conclusion

A rake receiver, which resolves the multipath signals corrupted by a fading channel, is the most complex and power consuming block of a modem chip. Therefore, it is essential to design a rake receiver be efficient in hardware and power. We investigated the design of a rake receiver for the WCDMA system, which is a third generation wireless communication system. Our rake receiver design is targeted for mobile units, in which low-power consumption is highly important. We made judicious judgments throughout our design process to reduce the overall circuit complexity by trading off with performance. The reduction of the circuit complexity results in low power dissipation for our rake receiver.

As the first step in the design of a rake receiver, we generated a software prototype in MATLAB. The prototype included a transmitter, a multipath Rayleigh fading channel as well as a rake receiver with four fingers. The MATLAB model for the rake receiver consisted of models for demodulators, a moving average block, a channel-frequency estimator, a compensator, a power estimator, a time tracker, and a combiner. Using the software prototype, we verified the functionality of all blocks of our rake receiver, estimated the performance in terms of bit error rate, and investigated trade-offs between hardware complexity and performance. The trade-off included truncation of output bits for three blocks. The demodulator outputs were truncated from 16 bits to 9 bits with the average performance degradation of 0.05 dB, and the channel-frequency estimator outputs from 10 bits to 9-bits with the average degradation of 0.13 dB. Finally, the compensator outputs were truncated from 19 bits to 15 bits with performance loss of 0.23 dB.

After the verification and design trade-offs were completed, we manually developed a rake receiver at the RT level in VHDL. The VHDL model included a frequency offset estimator, a clock generator, a control signal generator, a deskew-combiner, a SRAM controller, and an interface unit in addition to the functional blocks in the software prototype. A few schemes were incorporated into our rake receiver to enhance its performance. The deskew-combiner was a combination of a deskewer, a combiner, and

an SRAM controller. We employed two separate register blocks, instead of a rather brute force approach of using single block, to enable continuous deskew operations even in accidental cases. Also, the deskew-combiner provided three dump modes for the SRAM depending on the register block's status. The three different dump modes reduced the size of register blocks. In order to check the availability of the data for each finger output, we used a priority encoder instead of scanning the entire finger outputs, which saved hardware and resulted in reducing power dissipation.

As the final step, the RT level design was synthesized to gate level circuits targeting TSMC 0.18 μm CMOS technology under the supply voltage of 1.8 V. We estimated the performance of our rake receiver in area and power dissipation. Our experimental results show that the total power dissipation for our rake receiver is 56 mW and the equivalent NAND2 circuit complexity is 983,482. We observed that each finger consumed about 14.8 % of the total power of the rake receiver. Hence, the four fingers are responsible for 59 % of the power, and the deskew-combiner consumes the remaining 33.4 % of the total power. Each finger occupies 21.6 % of the total area and the deskew-combiner takes up 9.9 % of the area. The ratio of power consumption to area is less for fingers than for the deskew-combiner, as the average operating frequency of the fingers is four times slower than that for the deskew-combiner.

In summary, we investigated a rake receiver design for the WCDMA systems in this thesis. We believe that our rake receiver is efficient in area and power owing to judicious judgments in trading-off area and performance through the entire design process.

Glossaries

AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
CDMA	Code Division Multiple Access
CPICH	Common Pilot Channel
DPCCH	Dedicated Physical Control Channel
DPCH	Dedicated Physical Channel
DPDCH	Dedicated Physical Data Channel
EGC	Equal Gain Combining
FBI	Feedback Information
FDD	Frequency Division Duplex
FDMA	Frequency Division Multiple Access
FM	Frequency Modulation
GPS	Global Positioning System
GSM	Global System for Mobile Communication
IFFT	Inverse Fast Fourier Transform
IS-95	Interim-Standard 95
ISI	Inter-Symbol Interference
MAC	Medium Access Control
MRC	Maximal Ratio Combining
OVSF	Orthogonal Variable Spreading Factor
QPSK	Quadrature Phase Shift Keying
RRC	Root Raised Cosine
RTL	Register Transfer Level
SF	Spreading Factor
SIR	Signal to Interference Ratio
SNR	Signal to Noise Ratio
SSDT	Site Selection Diversity Transmission
TFCI	Transport Format Combination Indicator
TPC	Transmit Power Control

TDMA	Time Division Multiple Access
UE	User Equipment
WCDMA	Wide-band Code Division Multiple Access

References

[3G101] 3GPP Technical specification 25.101, UE Radio Transmission and Reception (FDD).

[3G201] 3GPP Technical specification 25.201, Physical layer – General description.

[3G211] 3GPP Technical specification 25.211, Physical channels and mapping of transport channels onto physical channels (FDD).

[3G213] 3GPP Technical specification 25.213, Spreading and modulation (FDD).

[3G214] 3GPP Technical specification 25.213, Spreading and modulation (FDD).

[Ber96] J. Bergmans, *Digital baseband transmission and recording*, Kluwer Academic Publisher, 1996.

[Cad01] "Low power option of Ambit BulidGates Synthesis and Cadence PKS", Product version 4.0.8, Cadence, May 2001.

[Cou02] L. Couch, *Digital and Analog Communication Systems*, Sixth edition, Prentice Hall, 2001.

[Din98] E. Dinan, B. Jabbari, "Spreading codes for direct sequence DCMA and wideband CDMA cellular networks," IEEE Communications Magazine, Sep. 1998, pp. 48-54.

[Gor02] P. C. Gorham, "CDMA2000 Benefits and Market Status", IMT-2000 Seminar, Lucent Technologies, 2002.

[Gue01] M. Guenachand, L. Vandendorpe, "Downlink performance analysis of a BPSK-based WCDMA using conventional rake receivers with channel estimation," IEEE Journal on selected areas in Communications, vol. 19, no. 11, Nov. 2001, pp. 2165-2176.

[Hol00] H. Holma, "Introduction to CDMA," Oct. 2000

[KTF03] http://www.ktf.com/eng/lab/mobile_imt2000/synchronize.jsp, "Asynchronous System VS Synchronous System", KTF Lab, 2003

[Kuo01] W. Kuo, "3G – 3.5G WCDMA Technologies, Markets, Products and the Future", Wiscom, 2001

[Neo00] K. Ryu, T. Chulajata, “High level design for 3GPP BS and its simulation results, version 1.0,” 2000.

[Oja98] T. Ojanpera, R. Prasad, *Wideband CDMA for Third Generation Mobile Communications*, Boston, MA: Artech, 1998.

[Pej01] M. Pejanovic, “On Optimization of 3G Cellular Systems Deployment”, ITU IMT-2000 Seminar, Oct. 2001.

[Pro00] J. Proakis, *Digital Communications*, Fourth Edition, McGraw-Hill, 2000.

[Qua01] “WCDMA Network Deployments: Asynchronous vs. Synchronous”, Qualcomm CDMA Technologies, 2001.

[Rap96] T. Rappaport, *Wireless Communications Principles and Practice*, Prentice Hall, 1996.

[Raz98] B. Razavi, *RF Microelectronics*, Prentice Hall, 1998.

[Roy00] K. Roy, S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*, John Wiley & Sons, 2000

[Smi75] J. Smith, “A computer generated multipath fading simulation for mobile radio,” IEEE Transactions on Vehicular Technology, vol. VT-24, no. 3, pp. 39-40, Aug. 1975.

[Tos00] H. Holma, A. Toskala, *WCDMA for UMTS Radio Access for Third Generation Mobile Communications*, Wiley, 2000.

[Vit95] A. Viterbi, *CDMA Principles of Spread Spectrum Communication*, Addison-Wesley, 1995.

[Yea01] G. K. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, 2001.

[Zie01] R. E. Ziemer, “3G CDMA – WCDMA and CDMA2000”, IEEE Communications Society Distinguished Lecture Program, May 2001.

Vita

Jina Kim was born in Seoul, Korea, on December 3, 1977. She received her Bachelor of Science degree in Computer engineering from Hanyang University, Seoul, in August 2000. She started in the M.S. program in Bradley Department of Electrical and Computer Engineering at Virginia Polytechnic Institute and State University from the fall of 2000. She worked under Dr. Dong S. Ha at the Virginia Tech VLSI for Telecommunications (VTVT) Laboratory and studied area and power efficient rake receiver for WCDMA systems. After receiving her M.S.E.E. Degree, she will continue as a Ph.D. student with Dr. Dong S. Ha from the spring of 2003.