

ECE 371NS PROJECT PHASE I HANDOUT

Instructor: Prof. N. R. Shanbhag

TA: Tao Long

Issued: 09/09/99; Due: 10/08/99

1 Introduction

The team project for ECE 371NS Fall 1999 is to design a RAKE receiver for a wireless DS-CDMA system. The project is divided into three phases to help you pace the progresses throughout the semester. During Phase I, you will learn the theory behind RAKE receivers and use MATLAB to simulate your design. The framework of the architecture will be provided. However, there will be enough details and approaches left for you to choose. Two test files will be provide to evaluate the performance of your design.

In Phase II, you will implement the algorithm in VHDL, synthesize it using SYNOPSIS, and verify the functionality. Besides the project handout, a short VHDL manual and a tutorial will be provided. In that manual, you will find examples and commonly used commands to bring you up to speed. At the end of this phase, you should have a fully functional and synthesizable VHDL model of your RAKE receiver.

In Phase III, you will use CADENCE automated place-and-route tools to generate the final layout. In this phase, the area constraint will be added to the specifications. If everything goes well, you should have a fabrication-ready layout at the end of the semester.

The purpose of this project is to provide you an opportunity to experience the entire design flow of a basic communication system. You will also have the chance to apply many of the techniques discussed in the lectures, and learn how to collaborate efficiently between you and your teammate. Last but not least, we hope you have some fun.

1.1 Getting Started

All the software used in this project are available on the EWS machines. If you do not have an account with the EWS yet, please contact a lab assistant or system administrator immediately. It is very important to have a working account from the beginning of the project.

Once you log on to your account, type `ece371`. This script will create your ECE 371NS home directory and copy some setup files. Once the script finishes, make a directory called `phase1`. Keep all the files you create for Phase 1 under this directory. Next, you need to copy two MATLAB binary files to your `phase1` directory. The files are:

```
~taolong/ece371.work/phase1/benchmark.mat  
~taolong/ece371.work/phase1/pdata.mat
```

1.2 Special note for Phase 1:

For some reason, MATLAB would not run after you execute `ece371` script. So after you log on, you should manually change to your ECE371NS work directory without running `ece371`, and then start MATLAB.

2 Preliminaries

2.1 Multipath Channels

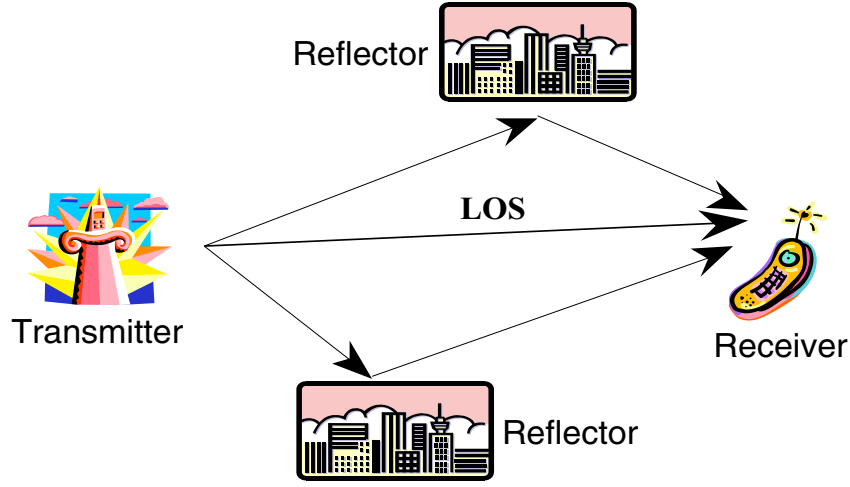


Figure 1: Typically multipath environment.

Echoes in multipath wireless communication channels are usually caused by reflective surfaces, such as buildings and automobiles. Echoes are essentially delayed and attenuated replicas of the original signal. Figure 1 illustrates a typical multipath environment. To keep things simple, we will assume the existence of a direct path between the transmitter and the receiver for this project. This path is called the line-of-sight (LOS), and is the shortest and strongest path. The reflection paths are longer than the LOS. Therefore, the reflected signals take longer time to reach the receiver. Besides the delay in a reflected signal, we should also expect some signal power loss due to the imperfect reflection as well as the longer distance. The third parameter of a signal path is phase distortion. All the signals we send are complex signals in nature. The phase shift caused by a multipath channel effectively rotates the signal coordinates in a complex plane about the origin. So, if 1 or $\exp(j0)$ is sent and the channel shifts the signal by $\frac{\pi}{4}$, then we receive j or $\exp(j\frac{\pi}{4})$. To sum things up, we obtain the following channel impulse response for multipath channels:

$$h(t) = \sum_{k=0}^{K-1} a_k \delta(t - t_k) \exp(j\theta_k)$$

where a_k is the k^{th} path gain (or loss), t_k is the k^{th} path delay, and θ_k is the k^{th} path phase shift.

Multipath channels are very common in wireless communications, and in reality things can get a bit tricky. For example, all the three parameters can change over time, and they may change rapidly or slowly depending on the environment. There are two kinds of environments—indoor and outdoor. In this project, we will only focus on indoor environments. Generally speaking, indoor environments have very short echo delays and sharp gain attenuations. These characteristics allow us to achieve a higher bit rate than in an outdoor multipath environment. Figure 2 shows a snapshot of the impulse amplitude response of such a channel.

2.2 Channel Specifications

The channel model we employ in this project is a much-simplified version of a multipath channel so that you do not have to come up with something patentable to get the project done. The channel is described

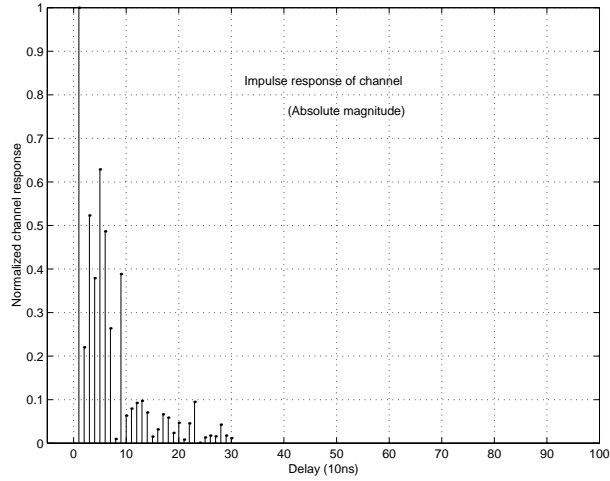


Figure 2: Indoor multipath channel impulse response.

as follows.

- Number of paths: ≤ 4 (1 LOS path, up to 3 reflective paths)
- Path gain: 0 dB for LOS, ≤ -6 dB (amplitude) for reflective paths, slow varying
- Path delays: 2 chips between two adjacent paths
- Phase distortions: random, slow varying
- Chip SNR: ≤ 9.54 dB

2.3 Rake Receivers

The essence of the RAKE receiver is a matched filter. A matched filter is a linear filter designed to provide the maximum SNR at its output for a given transmitted symbol waveform. Using Schwarz inequality, we can prove the following theorem.

Theorem 1 (Matched filter) ¹ Suppose input pulse $A_s(t)$ is received in additive Gaussian noise whose power spectral density $N(f)$ is nonzero at all f . The signal-to-noise power ratio S/N at the output of filter $g(t)$ satisfies

$$\frac{S}{N} \leq A^2 \int_{-\infty}^{\infty} \frac{|S(f)|^2}{N(f)} df$$

with equality achieved by the filter specified by

$$G(f) = \frac{S(f)}{N(f)}.$$

When the noise is white (e.g., AWGN), $N(f) = N_0/2$. Therefore, the filter has the same frequency response as the transmitted signal. The despreading process can be considered as matched filtering.

¹R.E. Blahut, "Theory of modems (class notes for ECE 361)," p45, Jan. 1999

RAKE receivers maximize SNR by matching the channel impulse response. Under ideal conditions (i.e., no interferences among echoes and the noise is white), the receiver SNR can be increased by a factor equal to:

$$SNR_{increase}(dB) = 10 \log \sum_k \left| \frac{a_k}{a_0} \right|^2$$

where a_0 is the gain of the LOS, and a_k is the gain of the k^{th} path. In reality, however, there are always interferences among echoes because the codes (PN sequence and Walsh sequence) we use have imperfect autocorrelations. So the SNR improvement may be lower than this ideal value.

2.4 DS-CDMA Base Station Transmitter

The base-station transmitter model is presented first to help you establish a complete view of the DS-CDMA system. It is important to understand the operations in transmitter since your RAKE receiver essentially performs the same operations but in the reversed order.

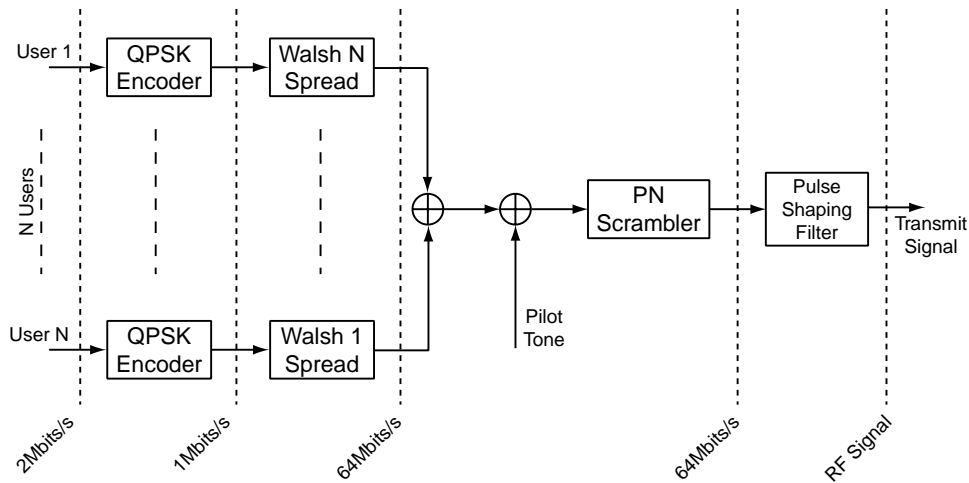


Figure 3: Base station transmitter block diagram.

Figure 3 shows the dataflow in a DS-CDMA base station transmitter. First, the user data is passed through a *quaternary phase-shift-keying* (QPSK) encoder. Both I- (in-phase) and Q- (quadrature) outputs of the QPSK encoder are spread by the same 64-bit Walsh sequence. After spreading, the I-signals of all users are summed and a pilot tone (an all-1 sequence) is added. Then the combined sequence is scrambled by a 1024-bit PN sequence. The same operations are done to the Q-signals. After pulse shaping and RF (radio frequency) modulation, the signal is sent to the physical channel.

2.5 QPSK Encoder

Antipodal signals ($\pm 1s$) are also known as *binary phase-shift-keying* (BPSK). This name comes from the phase interpretation in the complex plane. Note that 1 and -1 can also be written as $\exp(j0)$ and $\exp(j\pi/2)$, respectively. Therefore, mapping the unipolar binaries (0s and 1s) to antipodal signals is to encode binary bits with two phases (0 or π).

Quaternary phase-shift-keying (QPSK) is an extension of the BPSK. Instead of mapping one bit at a time, QPSK encoder maps two bits to one of the four quadrants of the complex plane. In a sense, BPSK only uses the real axis and QPSK consists of two BPSK orthogonal to each other. One of the BPSK is called the in-phase (I) component (the real axis) and the other is referred to as quadrature (Q) component

(the imaginary axis). As far as the operations are concerned, the QPSK encoder maps unipolar binaries to ± 1 , and then directs all the odd samples to the I-arm and all the even samples to the Q-arm.

At this point, you might wonder how to send a complex number. Recall that sine and cosine are mutually orthogonal. So we can modulate the real part using cosine function and the imaginary part using sine function. The two modulated signals are then summed and sent to the channel. The real part can be recovered by multiplying the same cosine function coherently to the received signal, and the imaginary part by the same sine function. (A lowpass filter might be required also.)

2.6 Raised Cosine Function And Square-Root Raised Cosine Function

The simplest pulse shape is the square pulse (binary values, 1 or 0). However, the frequency response of such pulse spreads out rather widely. This is very undesirable because of the strict restrictions on channel bandwidth usage wireless communication systems have to comply. Ideally we would like choose a pulse shape whose frequency response is band limited, e.g., *sinc* function. The problem is that *sinc* function extends from $-\infty$ to ∞ in the time domain and attenuates relatively slowly. This leads to high complexity DSP circuitry. As a compromise, raised cosine function and square-root raised cosine function are used.

Definition 1 (Raised cosine function) *Raised cosine function has the following Fourier transform:*

$$S(f) = \begin{cases} T & 0 \leq |f| \leq \frac{1-\alpha}{2T}, \\ \frac{T}{2} [1 - \sin(\frac{\pi T(|f| - 1/2T)}{\alpha})] & \frac{1-\alpha}{2T} \leq |f| \leq \frac{1+\alpha}{2T}, \\ 0 & |f| \geq \frac{1+\alpha}{2T} \end{cases}$$

where $\alpha = 2TW - 1$ is a parameter between 0 and 1.

The raised cosine function reduces to *sinc* function when $\alpha = 0$. This is also the smallest bandwidth a raised cosine function can span, and is used as a reference. It is easy to see that α indicates the excessive bandwidth usage compared to this reference bandwidth. The larger α is, the wider the bandwidth raised cosine function spans and the faster the function attenuates in the time domain. The inverse Fourier transform of the above function is

$$s(t) = \frac{\sin(\pi t/T)}{\pi t/T} \frac{\cos(\alpha \pi t/T)}{(1 - 4\alpha^2 t^2/T^2)}.$$

One important feature of the raised cosine pulse is that it satisfies Nyquist criterion, which guarantees zero intersymbol interference. Therefore, the bit period can be as short as T while the pulse main lobe width is $2T$.

Definition 2 (Nyquist criterion) *A Nyquist pulse for signaling interval T is a pulse $s(t)$ that satisfies*

$$S(f) = \begin{cases} 1 & f = 0 \\ 0 & f \neq 0 \end{cases}$$

Square-root raised cosine pulse frequency response is the square root of that of the raised cosine function. When convolving a square-root raised cosine function with itself in the time domain, we get the raised cosine function. Wireless communication systems often use square-root raised cosine pulse to shape each bits at the transmitter, and filter the received signal with the same pulse shape at the receiver to improves *SNR*. The idea is again matched filtering.

In this project, we use the square-root raised cosine function of $\alpha = 1$ as the pulse shape. In other words, the transmitter replaces each bit by such a pulse scaled to the bit value. Some common pulses and their frequency responses we have talked about are shown in Figure 4.

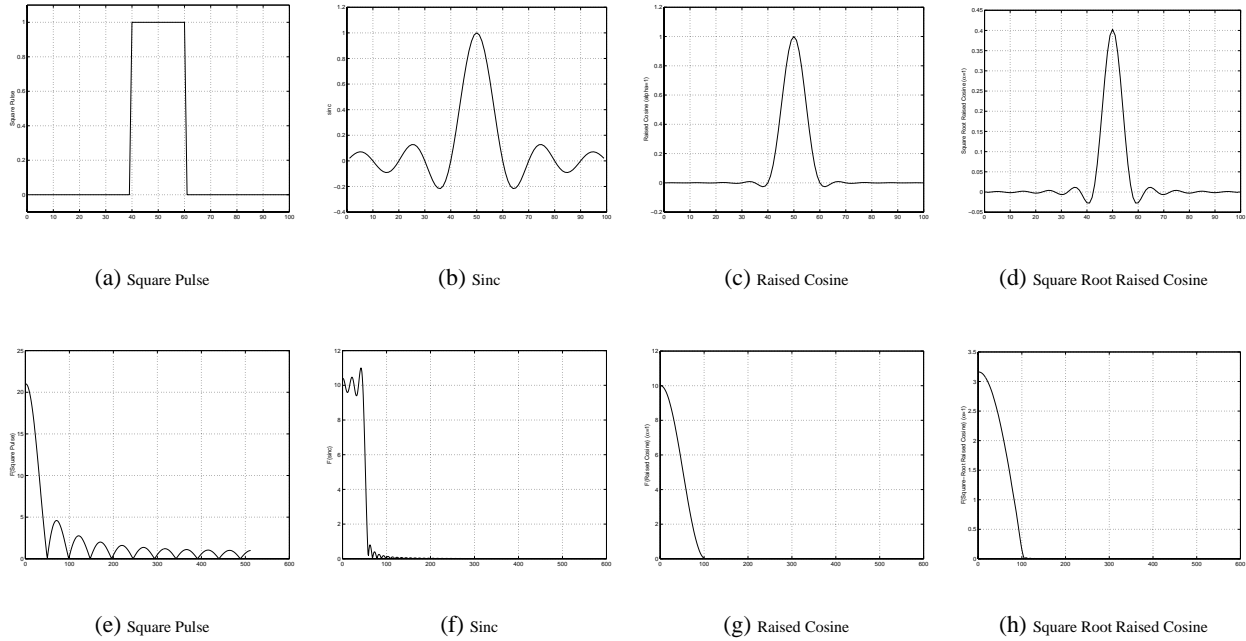


Figure 4: Different pulses. (a)–(d) are time response. (e)–(h) are frequency response.

2.7 Synchronization and Channel Estimation

The pilot tone makes synchronization and channel estimation possible at the receiver. The pilot tone can be considered as a special user in the system, except that this user sends only 1s at all times. The spreading sequence for this special user is a 64-bit all-1 sequence (or Walsh sequence with seed 0). Therefore, the scrambled the pilot tone is the PN sequence itself. Note that we use the same pilot tone for the I- and Q-signals.

A simple sliding correlator can be used to perform synchronization. The correlator tries all of the possible alignments between the PN sequence and the received signal. Because of the properties of PN sequence, the correlator output is very large when the two signals are synchronized. The pseudo-code for such algorithm is as follows:

```

SFLAG = 0;
reset PN generator;
for (; SFLAG == 0 ;) {
    ACCU = 0;

    for (CNTR = 0; CNTR <= N ; CNTR++)
        ACCU = ACCU + INPUT * PN;

    if (ACCU >= THRESHOLD)
        SFLAG = 1;
    else
        reset and stall PN generator for M clock cycles;
}

```

In the above pseudo-code, SFLAG indicates if synchronization has been achieved. ACCU is the accumu-

lator value, PN is the PN generator output and THRESHOLD is a predetermined constant. N and M are chosen so that the loop can cycle through all possible alignment positions.

Channel estimation is done after synchronization. Because both the I- and Q-signals of the pilot tone are all-1 sequences, we can compute the phase distortion and the path gain. Specifically, the pilot tone without distortion is $(1 + j)$ (real part is the I-signal and imaginary part is the Q-signal). After synchronization, descrambling and despreading, the receiver decodes the pilot tone as $(a + bj)$. Then the gain and phase distortion is proportional to $(a + bj)(1 - j)$, and the correction factor is the complex conjugate of this quantity.

3 Project Description

3.1 Receiver Specifications

- Pulse shape: square-root raised cosine pulse ($\alpha = 1$), main lobe width $2T_{chip}$.
- Analog front-end sampling period: $T_{chip}/2$
- Sync time: $\leq 5,000 T_{symbol}$
- Out-of-sync response time: $\leq 128 T_{symbol}$
- BER: $< 0.5\%$ using `benchmark.mat` input
- Symbol rate: 2 Mb/s
- Spreading sequence: 64-bit Walsh sequences (seed 1, 2, 3 for users, seed 0 for pilot tone)
- Number of users: 3 users and 1 pilot tone
- Modulation method: QPSK with mapping $[0, 1] \rightarrow [1, -1]$

3.2 Rake Receiver Architecture

A typical RAKE receiver architecture (Figure 5) includes three components—chip-matched filter, synchronization unit and data combine unit. The chip-matched filter is the same as the pulse shaping filter. This filter usually runs at a much faster clock rate compared to the rest of the circuit. This is to improve synchronization accuracy and to reduce the analog front-end design complexity. The synchronization unit is responsible for synchronizing the local PN and Walsh generator to the received signal, and provide channel impulse response estimation using the concepts we just developed in the previous section.

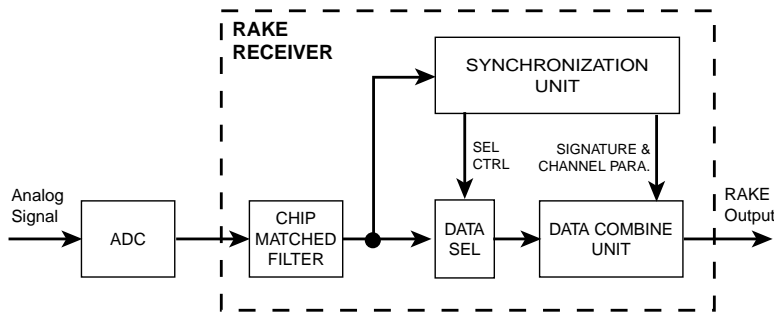


Figure 5: RAKE receiver block diagram.

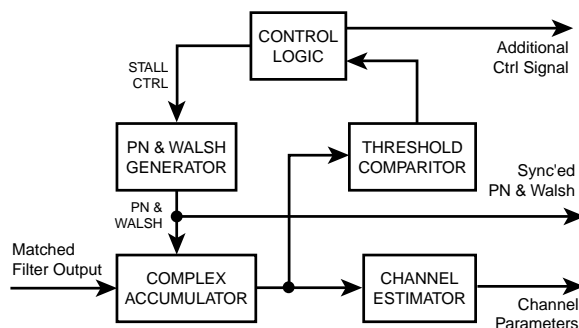


Figure 6: Synchronization unit block diagram

In this project, we will assume that the receiver analog front-end has already been designed. The analog front-end demodulates the RF signal and outputs analog samples of the I- and Q-signals at twice the chip rate. It is guaranteed that every other output is sampled at the pulse peak. In addition, an automatic-gain-control (AGC) circuitry can maintain the output power of this analog front-end at unity.

3.3 Analog-To-Digital Converter

In this project, we provide you the output from the analog front-end outputs (I- and Q-signals). You need to decide the precision for your receiver, and quantize the sampled analog signals accordingly. Your final receiver design should take the quantized inputs. (You only need to “implement” the ADC in MATLAB.)

Use the following model for ADC:

- Input range: $[-1,1]$ (because AGC circuitry keeps the power at 1)
- Number of bits: your choice
- Nonlinear distortion: 3^{rd} harmonic only, 0.4%
- Gain error: 1%
- Offset error: 2%

3.4 Chip-matched Filter Unit

This unit consists of two identical FIR squared-root raised cosine ($\alpha = 1$) filters. One is for the I-input and the other is for the Q-input from the ADCs. The filters operate at the same speed as the analog front-end sampling rate. Depending on the precision, the filter length may vary.

3.5 Synchronization Unit

This unit is probably the most complex part of this project. A block diagram is shown in Figure 6. This unit has two main functionalities: 1) to synchronize the local PN/Walsh generator with the received signal and 2) provide path parameter (gain and phase distortion) estimation. The theory of operations for both have been discussed in the previous sections. So here we only make some remarks.

- Sliding correlator is typically done with accumulators. For each possible alignment position, the accumulator has to accumulate for sufficiently many samples and then compare the result with

a preset threshold. Therefore, this is a very slow process. In order to meet the synchronization time requirement, you can either use multiple correlators to check multiple alignment positions in parallel, or reduce the number of accumulations for each alignment position. Regardless which approach you choose, the solution needs to be reliable enough (almost no false locks, and almost no miss locks). You are asked to present evidence in your report to support your choice.

- It is possible that the received signals becomes out of sync at any time. (For instance, the channel fluctuations may cause missing chips.) So your circuit should monitor the synchronization status even the local PN/Walsh generators have been synchronized previously. If such an event is detected, this unit should restart the synchronization process. The out-of-sync response time in the specs indicates the maximum delay between a detection and restarting synchronization.
- A hidden task for the synchronization circuitry is to generate the data-select control signal so that the correct half of the samples are fed to the data-combine unit. (Recall that the analog front-end outputs samples at twice the chip rate, so only half of the samples are sampled at the pulse peaks.)
- You may find that accurate channel estimation is a key to low BER. In this project, the channel varies very slowly. Therefore, you may want to apply running average to the channel estimations before passing them on to the data combine unit. You can use either an FIR or an IIR filter to perform running average. FIR filter produces accurate results, but is expensive in hardware implementation. IIR filter is small, but may not be as accurate.
- You can implement the two functionalities (synchronization and channel estimation) using separate hardware, or let them share the common components, e.g., accumulators. Keep in mind that the layout area is part of the design measure.

3.6 Data Combine Unit

Figure 7 illustrates the data combine unit architecture. It is capable of tracking four paths that are two-chip-delay apart. The hardware for each path is called one finger. Each finger has two multipliers. The first one multiplies the received signal with synchronized PN/Walsh sequence. The second multiplier corrects the phase distortion and path gain/loss. Both of the PN/Walsh sequence and the channel parameters are produced by the synchronization unit. At the output, symbols are coherently added and sliced. (The data slicer is not shown in the figure. The QPSK decoder is also omitted in the figure.) Note that you can tweak this architecture to make it more efficient (high speed, low power).

4 Project Grade

Design attributes (70%): sync time, BER, chip area, max speed;

Report and presentation (30%): style, clarity, team-work.

5 What To Do And What To Turn In

Two test files are provided. First, you will use `benchmark.mat` to study the BER and determine the precision of your design. In this file, each of the three users sends 20,000 random symbols. After QPSK encoding, the I- and Q-signals contain 10,000 symbols each. The first 5,000 symbols in the I- and Q-signals each are for synchronization. Use the second half of the symbols in the I- and Q-signals

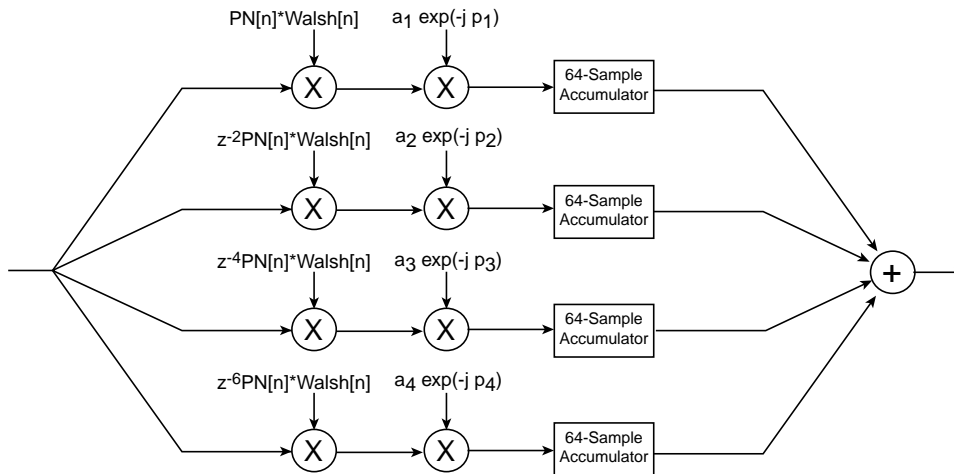


Figure 7: Data recovery block.

(5,000 from I-arm and 5,000 from Q-arm, i.e., 10,000 symbols total) to evaluate the BER. Determine the minimum precision based on the BER.

benchmark.mat

Variable	Data type	Description
di	single	analog front-end output of the I-signal (power = 1)
dq	single	analog front-end output of the I-signal (power = 1)
pn	int8	1024-bit PN sequence used for scrambling
sig1	int8	64-bit signatures for user 1.
usr1	int8	original user 1 data. Use this to determine the BER.

The PN sequence is generated from the primitive polynomial [4005] in octal representation. Only the first 1024 bits are used. The signature sequences are 64-bit Walsh sequences with seed 1, 2 and 3 for user 1, 2 and 3, respectively. (Seed 0 is reserved for the pilot tone.) Even though these sequences are provided to you, your design should include the PN and Walsh sequence generator.

The second file (pdata.mat) is similar to benchmark.mat, except that it contains twice as many data and the original symbols are not provided. Use the receiver program you developed for benchmark.mat to decode all three users' messages. Name the last 20,000 decoded symbols from each user user1, user2, and user3, respectively. A script will be posted on the project web page to present the decoded symbols in an intuitive manner.

pdata.mat

Variable	Data type	Description
di	single	Analog front-end output of the I-signal (power = 1)
dq	single	Analog front-end output of the I-signal (power = 1)
pn	int8	1024-bit PN sequence used for scrambling.
sig	int8	Signatures for all three users.

You may have noticed that the data types of the variables are other than double. They are chosen to reduce the storage space without changing the values. MATLAB can only manipulate double precision

data. So to use these two files, you need to first convert the variables to `double` type. This can be done, for instance, with the following command:

```
>> di = double(di);
```

We only provide you the interfaces for `rake()` and `adc()` functions. You can decide the function interfaces for all other functions or subroutines. (The idea is that with the two standard interfaces anyone should be able to test your algorithm. Of course, they have to provide proper analog front-end outputs.) The analog front-end output will be processed by `adc()` routine first. The output of the `adc()` routine will then be processed by `rake()` routine to produce the unsliced output.

Function: `adc()`

<i>Interface:</i>	<code>result=rake(data,nbits,g_err,h_err,os)</code>
	<code>data</code> data to be quantized (arbitrary-sized real matrix)
	<code>nbits</code> number of bits
<i>Inputs:</i>	<code>g_err</code> gain error (percentage, from 0 to 1)
	<code>h_err</code> 3^{rd} order harmonic error (percentage, from 0 to 1)
	<code>os</code> offset error (percentage, from 0 to 1)
<i>Outputs:</i>	<code>result</code> ADC output, same size as the input

Function: `rake()`

<i>Interface:</i>	<code>result=rake(di,dq,pn,sig)</code>
	<code>di</code> quantized I-signal
	<code>dq</code> quantized Q-signal
<i>Inputs:</i>	<code>pn</code> PN sequence
	<code>sig</code> user signature
<i>Outputs:</i>	<code>result</code> unsliced output of the data-combine unit. It should be a column vector.

You need to turn in the following:

- the complete pseudo-code for your algorithm (including all the parameters)
- a complete copy of your MATLAB code
- the histogram of the RAKE receiver output for `benchmar.mat` data (last 10,000 symbols before slicing)
- the “intuitive” representation for the decoded `pdata.mat`.
- a report of the BER, precision and how you choose all the parameters in your design. (Some of you may decide to increase the precision to reduce the BER, or make synchronization parallel to speed up the sync time. We want to know how you come up with the decisions.)
- Finally, tar and gzip your MATLAB code as well as the unsliced output of your RAKE receiver simulation for both given data files. Email the gzipped file to the TA (taolong@uiuc.edu). Make sure to write up a README file to explain the functionality of each MATLAB routine and how to examine your simulation results.

As a reminder, please keep an eye on the course web page for updates and hints.