

# **Design and Implementation of Power Line Carrier Communication System**

## **Preface**

This is the report of a project we conducted at National Key Laboratory of Communication since September, 2013. The report is all by myself while part of the work was done by my roommate, Liu Dongzhu. We worked together at night and on weekends, under the supervision of Dr. Hu Jianhao, an insightful professor who is the Deputy Director of the laboratory.

During the whole process of researching, I learned how to think and organize, how to solve problems in the study. There were plenty of innovative ideas bursting out. Although most of them did not turn out feasible in the verification concerning resources and efficiency, the exploring experience will make me stronger in future researches and closer to the cutting edge.

This report records my thoughts and progress in researching. It is not designed to publish, some details are not eliminated, as I want not only to present what we do, but also show the logic connections between different parts. Namely, to indicate the deficiency of the current system and point out the reason why we have to add another part.

If there is something wrong or inappropriate in the report, please tell me. My e-mail is liubeibei789@gmail.com. Your comments and criticism are of greatly importance to me. Thanks a lot !

## **Abstract**

It is a tricky problem to measure electricity consumption in factories and plants. The difficulty lies in the high-intensive and complicated interference in the environment. Measuring through the common communication system will generate great error. A power line carrier communication system is designed for the purpose of high reliability data transmission under low signal to noise ratio (SNR) condition. Power-line communications system operate by adding a modulated carrier signal to the wiring system. Different types of power-line communication use different frequency bands. The system is based on Code Division Multiple Access (CDMA), adopting the spread spectrum technology. The ultimate hardware implementation is a Plug-and-Play device applied to the wire plug.

## **Overall design**

The hardware platform we use is a board integrated with DSP and FPGA. (see picture below). As DSP and FPGA has its own strength and weakness, we separate the task into two parts. Make interleaving as the watershed. Starting from the interleaving part of transmitter to the

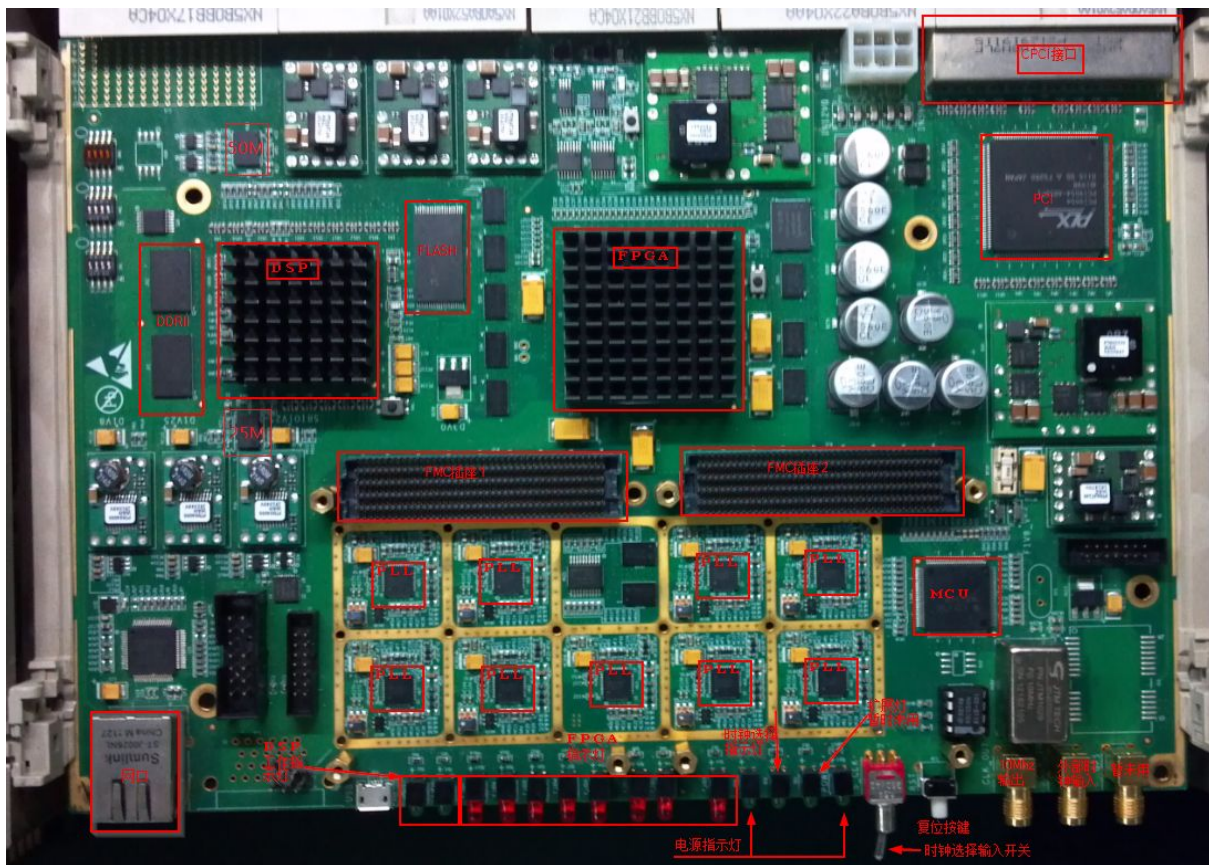
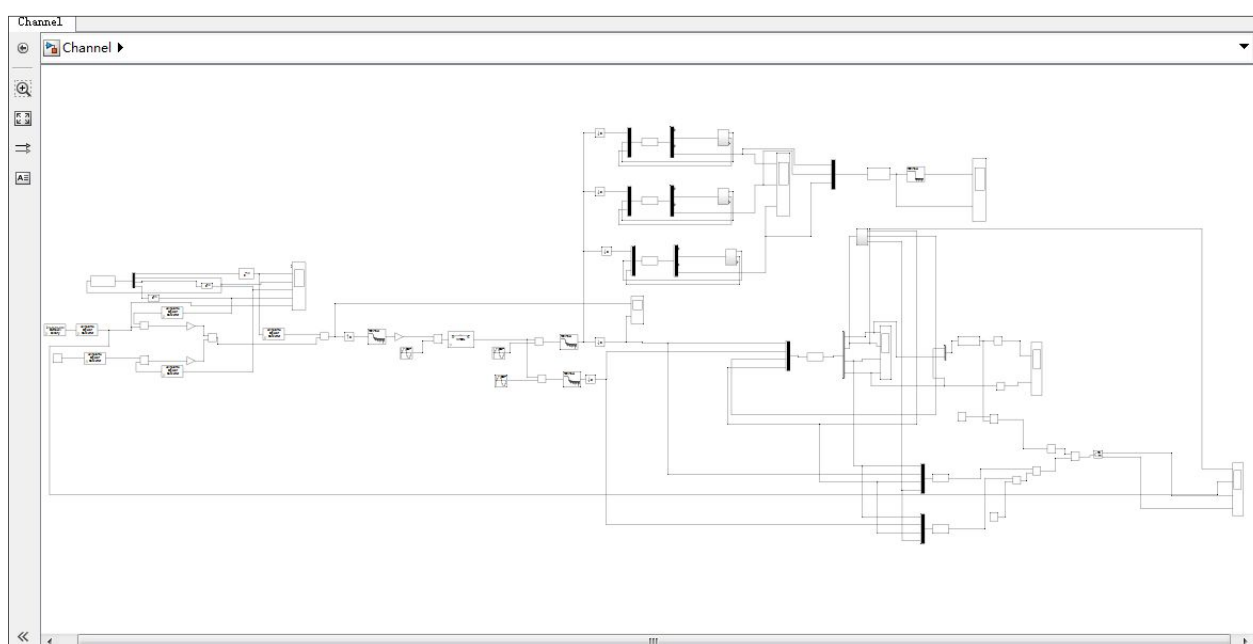
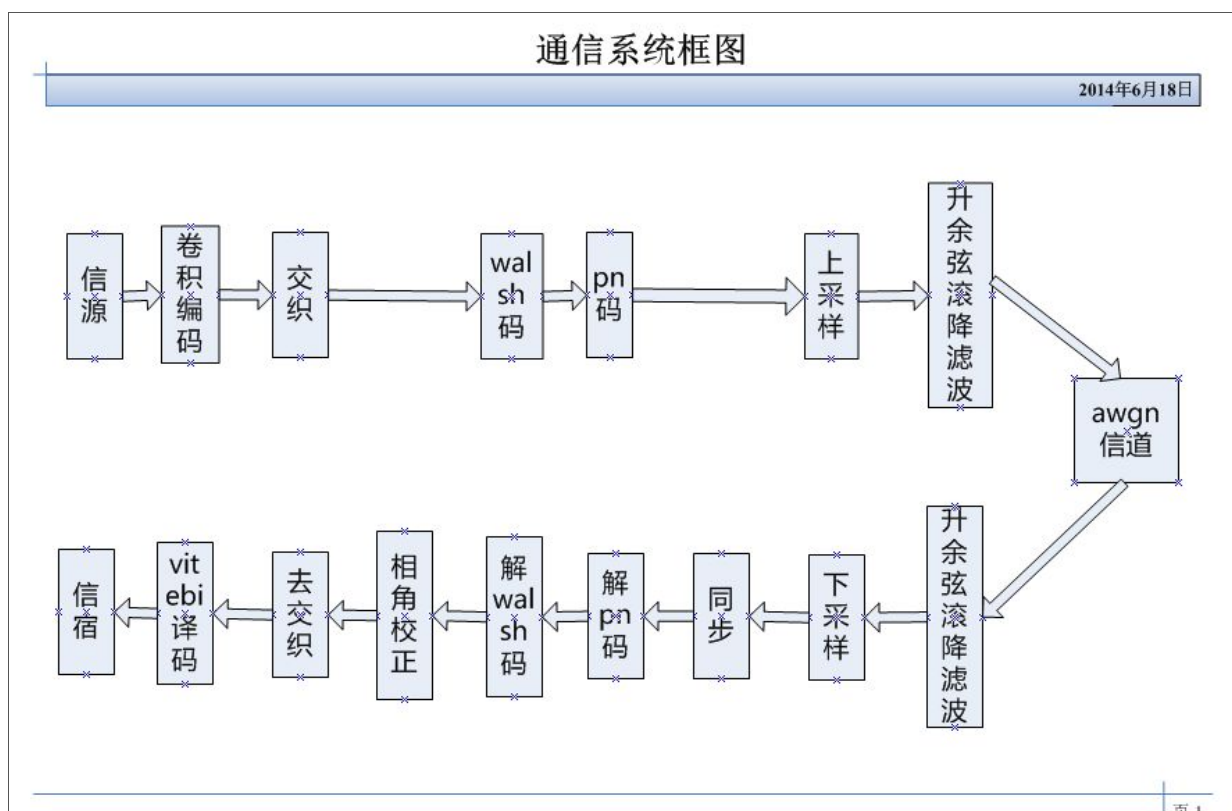


Figure: block diagram of overall design



### Specific design and simulations

Before implementing the design in hardware, we do simulation in software to test its correctness and reliability. Moreover, the simulation data can be used in the validation of system's output. We use Matlab to assist our design.

### Source coding ( decoding in receiver)

There are two major steps in source coding:

(1) Transforming the analog signal to digital signal, namely A/D convert.

Information in real world usually presented in analog form. If we want to transmitted it in digital system, which has many advantages over analog system, we must first transform the analog signals into digital signals. How we do that ? We use different combination of digital impulse (0 or 1) to represent different amplitudes of the analog signal. To raise the efficiency, we want to represent a given analog signal with as few digits as possible, on the premise of the transmission quality requirement is met.

In this project we leave out the collection and coding of analog signals, starting from the practical physical channel, that is, the original data is in the form of 0/1 sequence. Because the function of our system is to transmit data collection a certain electricity measurement device. We do not have to care about that process, the outcome of which is already digital. Instead, we just focus on the process of transmission.

When applied to hardware, input data is read from computer in “txt file” form. In simulation, source signal is simulated by Bernoulli binary stochastic series.

(2) Pack the data into standard packages called frames

Format of the frame is determined while designing the system. We design the frame length to be 20 ms. It is under the bit rate of 9.6kbps. Therefore, the total bits per frame is  $9.6\text{kbps} \times 20\text{ms} = 192\text{bits}$ .

The 192 bits comprise of 32 bits of head, 144 bits of message, 8 bits of CRC and 8 bits of tail.

The message is as follows: ( In actual application, message is the data generated by certain device like electricity measurement device. Message below is by assigned by myself for simulation purpose.)

0111001100001100101001100111100010011001000100110101011100100111000101001100101001100100101100010011011000100100101011000100111001100001100101001100

The last 0 is the ‘\0’ for char string.

The 32 bit head is the pilot code, containing users’ information and synchronous information. The 8 bit Cyclic Redundancy Check (CRC) is generated by a program in C code. 8 bit tail is designed because when we complete the coding of a given block, the register need to be initialized to all zero. So for the convolution coder with constraint length of 9 (i.e. 8 registers in series), 8 zeros are supposed to attach to the data block. Besides, when conducting Viterbi decoding, we would be able to go back to the first row of mesh only if the last 8 bits are all zero.

Data fed into convolution coder is:

1011101010110101010010101000111001110011000011001010011001111000100110010001001  
1010101110010011100010100110010100110010110001001101100010010010101100010011100  
110000110010100110000100100000000 (192bits in total)

### 3.1 Convolution coding ( one kind of channel coding)

When transmitting in channel, digital signals may distort due to noise. Our goal is to raise the reliability, reducing the number of errors to the least. A checking mechanism is established therefore. On the transmitting side, we generate a checking code out of the message, according to a certain rule. This code is attached to the original code as a tail. On the receiving side, since it knows the rule for checking code, it can repeat the same process to test if there is any error occurs. If the rule is properly designed, it can even correct some bits of error.

It may seem confused to minimize the length in source coding while to increase the length in channel coding. Contradictory as it seems, actually, it embodies the trade off in the two competing indicators of efficiency and accuracy. In source coding, we tend to reduce the redundancy; in channel coding, we tend to raise the reliability at the cost of some redundant bits. The balance is reached depending on the practical situation.

Channel coding is also called error control coding. A coding scheme is actually a mapping rule, which converts from a small information space to a larger information space. For example, what we adopt in this project is (2,1,9) convolution coding. (2,1,9) means mapping every 1 symbol to 2 symbols, with constraint length of 9 (i.e.8 registers).

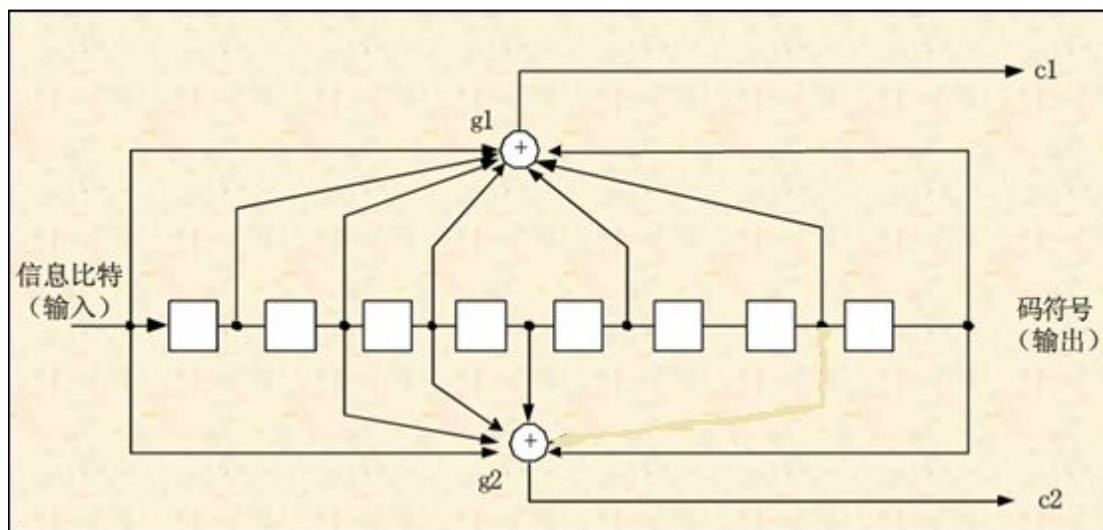
By this technique, we are able to reduce error rate largely. Assuming we have the same probability of error  $P$  in every single bit. If we use 1 symbol to represent 1 amount of message we want to convey, the ultimate probability of error is  $P$  ( $P < 1$ ). However, if we map that symbol to a larger space, say 2 symbols representing 1 amount of message, we have 00,01,10,11, four combinations in total. Only 2 of them hold meaning.(lets assume 00 and 11 are meaningful). Then the only case we may make mistake is the two bits both in error. That is a probability of  $P * P$ , which is much smaller than  $P$  in most cases.

For the block coding, the current  $n$  elements are only related to the current segment. The convolution coding is different, however, the  $n$  elements of the current segment of convolution code do not only related to the elements in current segment, but also related to the former  $n-1$  segments. There are  $nN$  symbols has certain relationship with each other. The performance of error detection improves as the increase of  $N$ .

Error correction performance of the convolution code improves as  $N$  grows, error rate greatly reduced. Convolution code has better performance than block code, on condition that the orders have the same order.

The generator polynomial determines the structure of convolution coder. When designing a specific communication system, designers are supposed to refer to corresponding standard to figure out the regulation of structure of convolution coder. (see figure below)

Figure: structure of the (2,1,9) convolution coder stated in communication standard IS-95



The generator polynomial for (2,1,9) coder are:

$$G_1 = 1 + D^1 + D^2 + D^3 + D^5 + D^7 + D^8 \quad G_2 = 1 + D^2 + D^3 + D^4 + D^8$$

S-function are as follows:



```

1  function [sys,x0,str,ts,simStateCompliance] = juanjibianma(t,x,u,flag) % (2,1,9) 卷积编码
15
16  function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
17  sizes = simsizes;
18  sizes.NumContStates = 0;
19  sizes.NumDiscStates = 9; %长度为7的移位寄存器
20  sizes.NumOutputs = 2; % 2个模二加法器
21  sizes.NumInputs = 1; %1个输入
22  sizes.DirFeedthrough = 0;
23  sizes.NumSampleTimes = 1; % at least one sample time is needed
24  sys = simsizes(sizes);
25
26  x0 = [0 0 0 0 0 0 0 0 0];
27  str = [];
28  ts = [1/9600 0]; % 速率为9.6kbps
29
30  simStateCompliance = 'UnknownSimState';
31
32  function sys=mdlUpdate(t,x,u)
33  sys(1) = u;
34  for i = 2:9
35  sys(i) = x(i-1); %依次向右移一位
36  end
37
38  function sys=mdlOutputs(t,x,u)
39  sys(1) = mod((x(1) + x(2)+x(3) +x(4)+x(6)+x(8)+x(9)),2); %生成多项式G1=1+D1+D2+D3+D5+D7+D8
40  sys(2) = mod((x(1) + x(3)+x(4)+x(5)+x(9)),2); % G2=1+D2+D3+D4+D8
41

```

The output of the convolution coding s-function is a 1\*2 vector, two elements are output of two module-2-adders respectively. An unbuffer block is set to combine the two series into one, which is the final output of convolution coder. (see figure below)

Figure :convolution coder

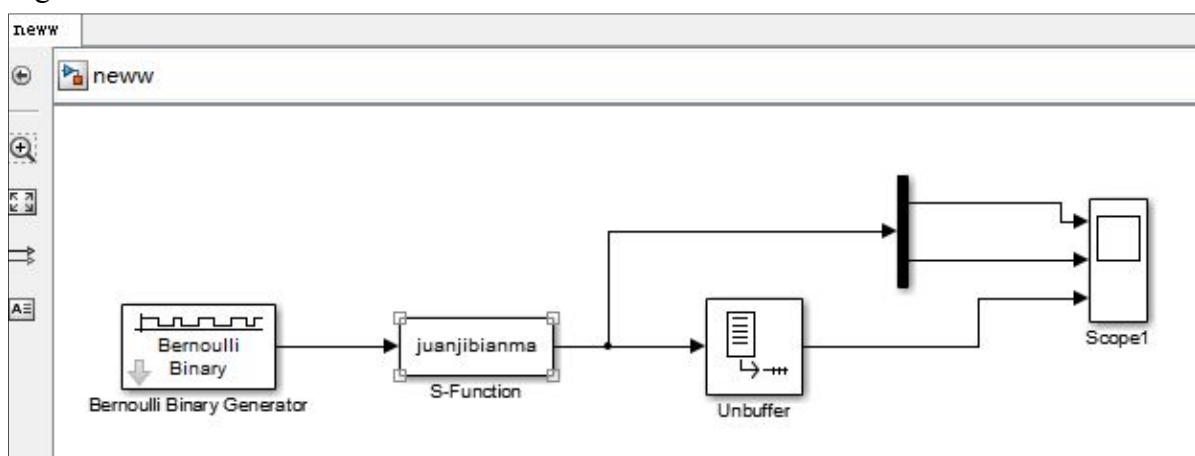
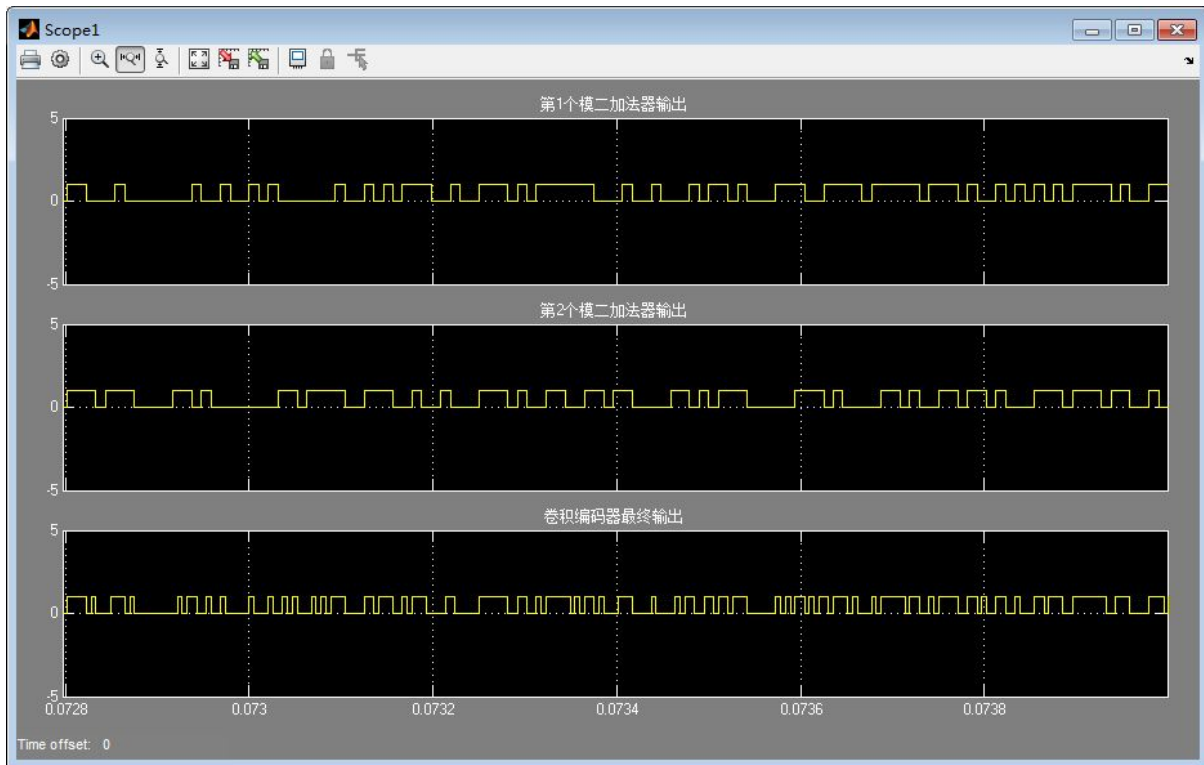


Figure : output of convolution coder



Data coming out of the convolution coder is written into txt file directly.

```
111000101100100000010001111110100111001000000101100101101010100101110101100001
0010111000111001101011011110100011111010011100000100110011111001110010111111101
1111110001111010010100110110101111011110010010001100001111110101111010001111101
0000101101011010111111101011000011101010001000101011111000001101101011100101100
100101110001110011010110111101000111000001001011110111110100101100
```

The three commonly used methods for decoding are algebraic decoding, Viterbi decoding and sequential decoding. In this project we use Viterbi decoding.

### Interleaving

The error detection and correction mechanism can only manage a certain length of bit error. Therefore, we should avoid too many bits making mistake in succession, which occurs in a lasting interference. A prevailing practice is to interleave. (reverse-interleaving in the receiver) Interleaving is a sort of time diversity, because a package of message is delivered in several time intervals.

There are two major methods of interleaving: (1) block interleaving (2) convolution interleaving. In this project we adopt block interleaving. Because block interleaving has already met the requirement and it is easier to program and more reliable than convolution interleaving does.



Specific method is : write by rows, read by columns. That is, when writing, search the matrix row by row, then element by element in each row; when reading, the contrary.

As input sequence will not stop to wait for the output sequence to be read, there must be two identical interleavers to write and read in turn, thus maintaining the continuity of transmission. The depth of interleaving is 80 ms, which is concatenated of 3 frames.

#### Unipolar to Bipolar Convert

After interleaving, we convert 0 and 1 to 1 and -1. This unipolar to bipolar convert is owing to the relationship of module-2-addition and multiplication. We tend to add the two signals then module two. In Simulink, we do the multiplication instead for the same effect.

Chart: relationship between module-2-addition and multiplication

Channel 1	Channel 2	module-2 -addition	module-2-additi on(logic error)	Channel 1	Channel 2	multiplication
0	0	0	0	1 (0)	1 (0)	1 (0)
0	1	1	0	1 (0)	-1 (1)	-1 (1)
1	0	1	0	-1 (1)	1 (0)	-1 (1)
1	1	0	1	-1 (1)	-1 (1)	1 (0)

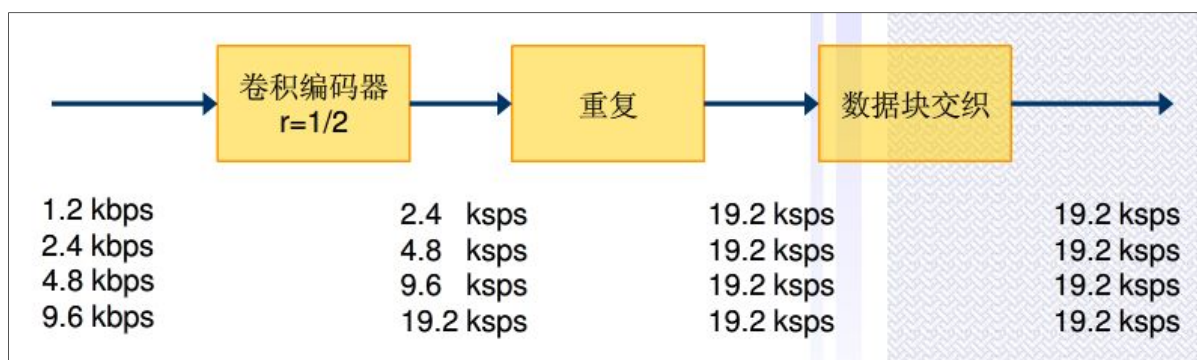
On the left is the original 0, 1 sequence. On the right is the mapped 1 and -1 correspondingly. Thereby, the multiplication in the new logic is exactly in accordance with the module-2-addition in the old logic. If mapping 0 to -1 while mapping 1 to 1, the result is exactly the reverse. Furthermore, if omitting the unipolar to bipolar convert, just multiply them will cause entirely logic failure.

#### Synchronization of speeds

In practical application, there are many channels with varies data speeds. If the speed of a certain channel is not at 9.6kbps, but is at a lower speed, then after convolution coding, it requires to do the repetition of symbol. It repeats each symbol for some times to enable the symbol rate reach 19.2kbps after convolution coding. Repetition of symbol keeps the bits fed into interleaver to be at a consistent speed.

In this simulation we just have two channels, one data signal and one pilot channel with all zeros. Speed difference does not exist. Therefore repetition is skipped.

Figure : speed changes in synchronization



Channel coding

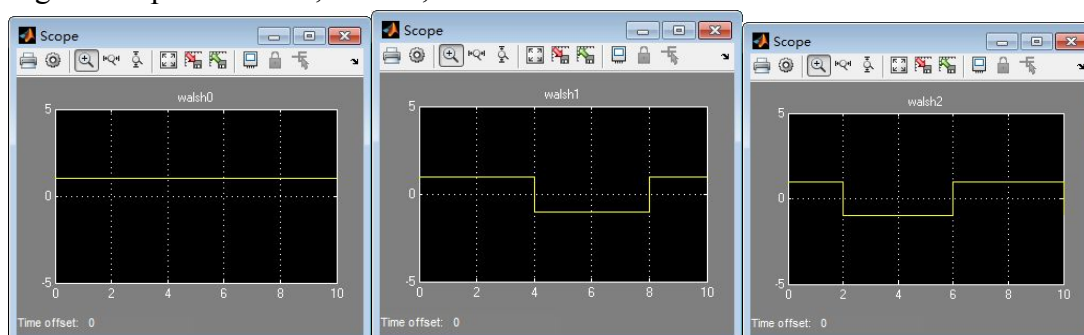
Spread spectrum modulation (demodulation)

Each logic channel has its corresponding spread spectrum code. after multiplication, it is mapped to the physical channel with 0 and 1.

In wireless communication standard IS-95, forward channel consists of four logic channels : 1 pilot channel ( physical channel : walsh0), 1 synchronous channel ( physical channel : walsh 32), 7 paging channels( walsh1~walsh7)and 55 traffic channels ( walsh8~walsh31, walsh33~walsh63).

Our system implements two channels. One data channel ( spread spectrum by walsh1) and one pilot channel with all logic zeros( spread spectrum by walsh2). Walsh0 is not applicable since it has 32 zeros in succession and 32 ones in succession. When the pilot multiply with walsh0, power spectrum is uneven. Symbol rate is also reduced after spread spectrum. Thus, it will weaken the effect of spread spectrum.(Carried to an extreme, if there are all 0 or all 1 in spread spectrum code, it is the same effect as not conducting spread spectrum at all).

Figure: shape of walsh0, walsh1, walsh2 code



The pilot is to restore the phase, therefore it should have higher power than that of the data. In simulation, we assign the two channels with different weight by adding block “gain” on two channels, 0.3 for data channel and 0.7 for pilot channel. Then add them up.

Common methods for spreading spectrum are: Direct Sequence Spread Spectrum (DS), frequency hopping (FH), time hopping (TH) and Chirp. We adopt Direct Sequence Spread Spectrum in the project.

Spread spectrum can be processed in two steps of modulation:

Step 1: spread spectrum modulation

Data sequence does the module-2-addition with spread spectrum code. In most cases, symbol rate of them are different. Therefore the data symbol should be spread in time domain to match the spread spectrum code. The multiple to spread depends on the ratio of symbol rate of spread spectrum code to the symbol rate of data rate. In this project, it is 64.

Step 2: carrier modulation:

Carrier modulation spreads the band to a large scope. It typically adopts constant amplitude modulation like BPSK, QPSK, with relatively low order. Because lower order has lower error rate, ensuring the reliability. Some higher order modulation like 8PSK and 16PSK, has higher efficiency. However, the reliability is not good.

Under the direction of the supervisor, we choose generator polynomials with a good properties of orthogonality. That is, it has a significant peak in autocorrelation and very clean in cross-correlation.

$PN=1+x^5+x^7+x^8+x^9+x^{13}+x^{15}$  or  $PN=1+x^3+x^4+x^5+x^6+x^{10}+x^{11}+x^{12}+x^{15}$

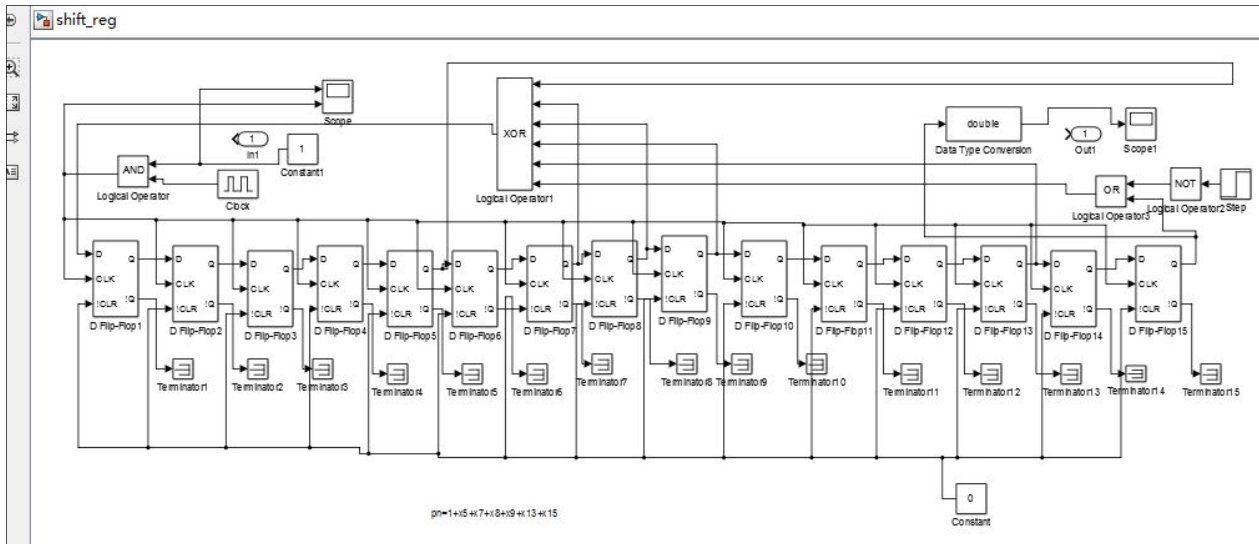
We use walsh1 and walsh16(or walsh32) for data and pilot respectively.

Sorting the PN generator polynomials in descending powers, expressed in vector as [15 13 9 8 7 5 1], namely [1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1]. Initial state should not be all zeros. All the other cases are applicable just different in time consumed to get the correlation peak. We take [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] as initial state in this system.

There is an alternative method, do the spread spectrum after the band-modulation. In that case, it is the modulated signal ( in 1.2288 MHz) that to multiple with spread spectrum code ( namely PN code, in 1.2288Mbps. Actually, this system is linear, the sequence of the blocks do not affect the outcome.

Considering that in the early-late-gate, we need to elicit data from two adjacent bits , we are supposed to have a disassembled shift register. Therefore we do not use the “pn generator” block in Simulink. Instead, we construct a shift register with flip-flops.

Figure : 15 order shift register with D-flip-flops



When considering the synchronization on receiver, it is better to make the PN and walsh align at the very beginning. So that when it comes to the acquisition, we can just move 64 symbols each time and do correlation, finding the correlation peak.

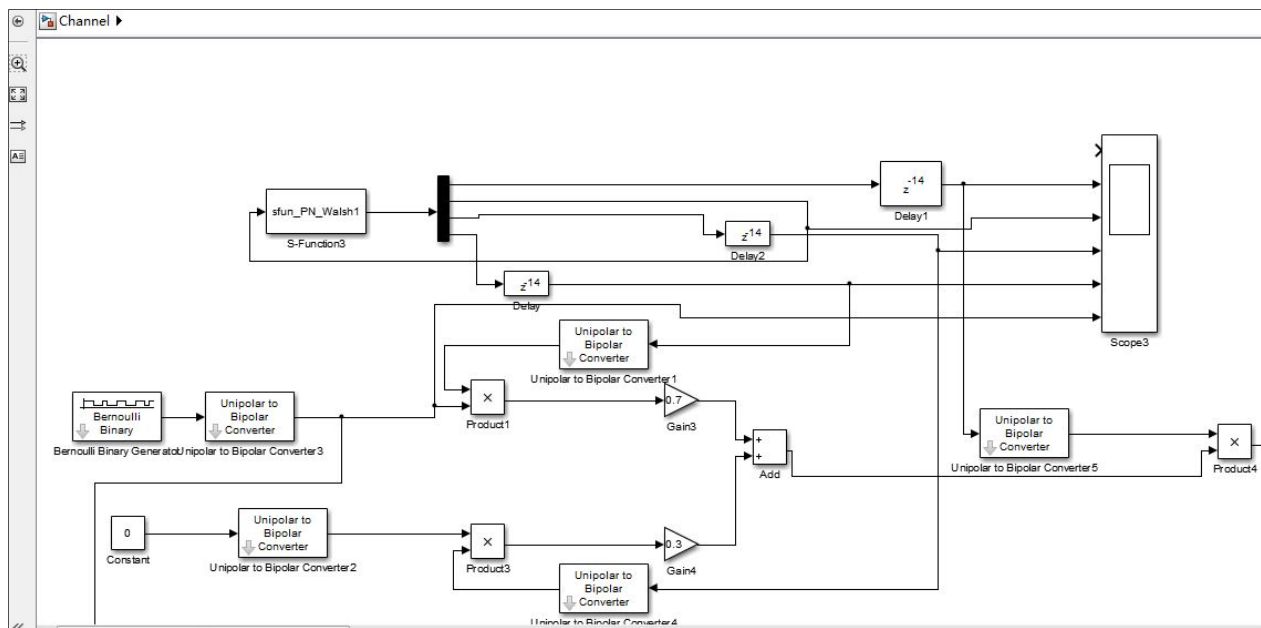
A salient feature for PN code is that there exists 14 zeros in succession, every 32767 bits a cycle. Thanks to this feature, we can search the sequence to find this sign“14 zeros”. When this sign occurs, we set it as a beginning. From this moment the walsh starts to send. In this way, the two sequences would align.

Since the cycle of PN is 32767, it should be add one bit to reach 32768 so that is divisible by 64 (i.e. Length of walsh), allowing it to be align all the time. Therefore, what we should do is to exact position of the 14 zeros, forcing the input PN to hold on a clock pulse, sending a 0 instead. This method is fair, but it has many output signals as indicating signals.

A modified method is to write the code a 15-order register in s-function, then put the 64 bit walsh directly into the s-function code. In this way the synchronization can be controlled directly by programming.

See how PN and walsh generator is incorporated into the system below.

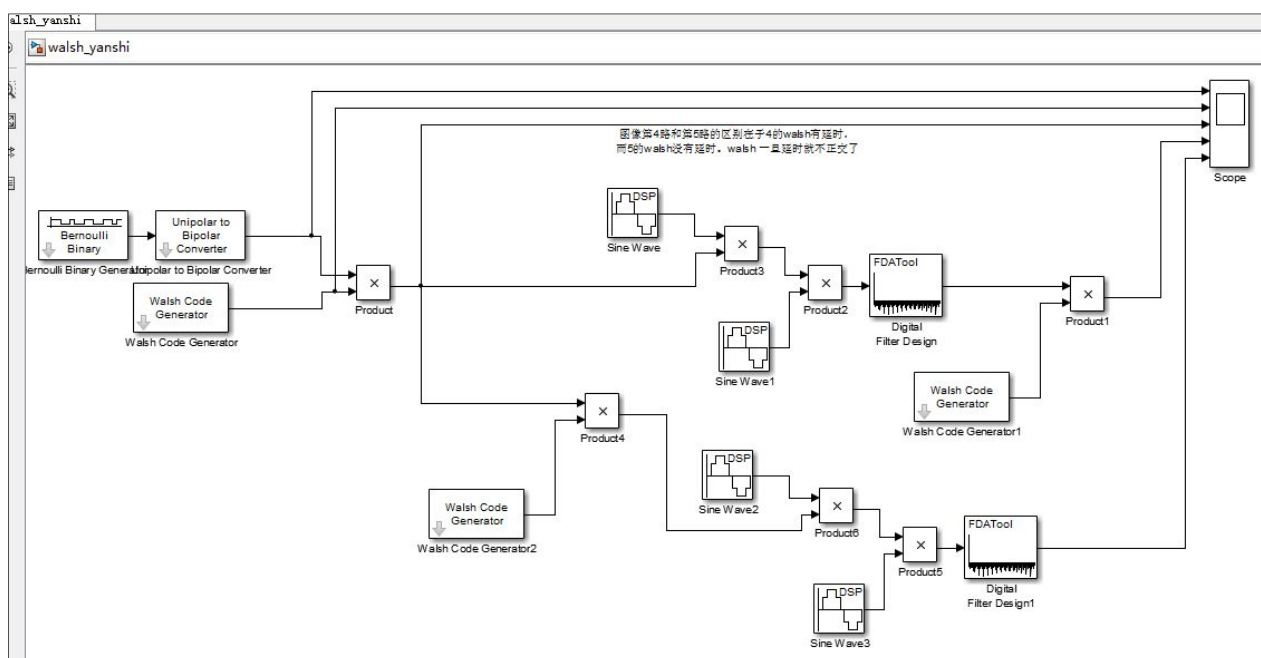
Figure: block diagram of PN and walsh generator in Simulink



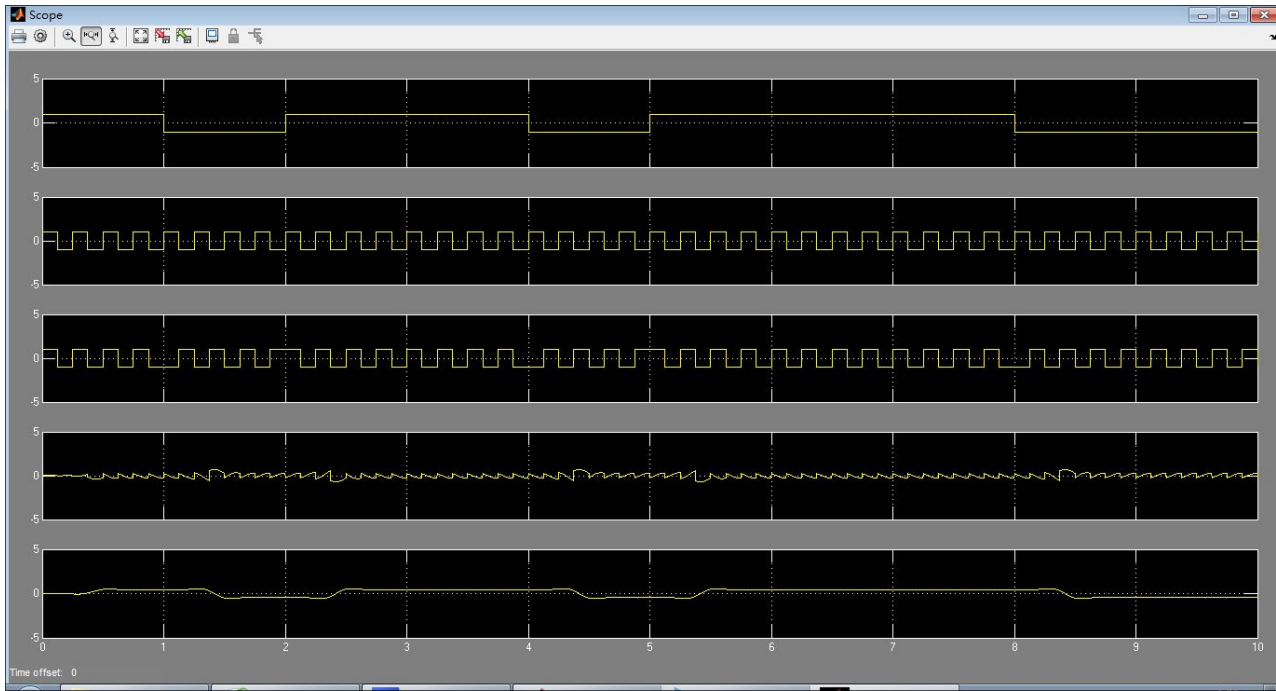
To test the orthogonality of walsh code, as well as the effect of delay on walsh, we have the block diagram in simulink below:

On transmitting side, Bernoulli sequence is to simulate the generated data.

The upper branch is the normal process of band-modulation and demodulation. On receiver it is de-spreaded by walsh of the same index. The lower branch is the to change the sequence of de-spreading and demodulation, namely do the de-spreading first.



The result is presented in the figure below. The five sub figures are : (1): original Bernoulli sequence 1011011100; (2): 8-order-walsh code with index 7; (3): the multiplication of Bernoulli sequence and walsh; (4) the upper branch (in normal sequence); (5): the lower branch (in changed sequence).



Walsh spread spectrum(de-spread spectrum) and band-modulation(demodulation) are both linear transform, which should not have care about the change of sequence. However, the fourth sub figure fails to restore the original data, while the fifth sub figure is able to restore the original data.

The difference lies in that the spread and de-spread of spectrum cause delay on the receiver. Therefore, the walsh code on receiver is not exactly in alignment with that on transmitter. The orthogonality is broken if two sequences are not align.

To solve this problem, we have to figure out the delay and take some measurement to compensate it. One solution is to let local walsh be shifted in advance, waiting for the exact moment to guarantee the orthogonality of the two walsh sequences. Now the problem is transformed into : how to find the delay from spread spectrum to de-spread spectrum.

s-function of the PN and walsh code generator is shown below.





```

67 -     shift_regist(1)=shift_regist(1);
68 - end
69
70 - if(shift_regist(15)==0)
71 -     cnt=cnt+1;
72 - else
73 -     cnt=0;
74 - end;
75
76 - if(cnt==14) % 计数14个连零，若发现14个连零则out输出为0
77 -     out=0;
78 - else
79 -     out=1;
80 - end;
81
82 - if(u(1)==0) % 如果输入为0，则选择walsh61
83 -     index=61;
84 - else
85 -     if(index==63) % 64阶walsh码的下标循环
86 -         index=0;
87 -     else
88 -         index=index+1;
89 -     end;
90 - end;
91
92 - sys =[ shift_regist(1) shift_regist(2) shift_regist(3) shift_regist(4) shift_regist(5) shift_regist(6) shift_regist(7) shift_regist(8)

```

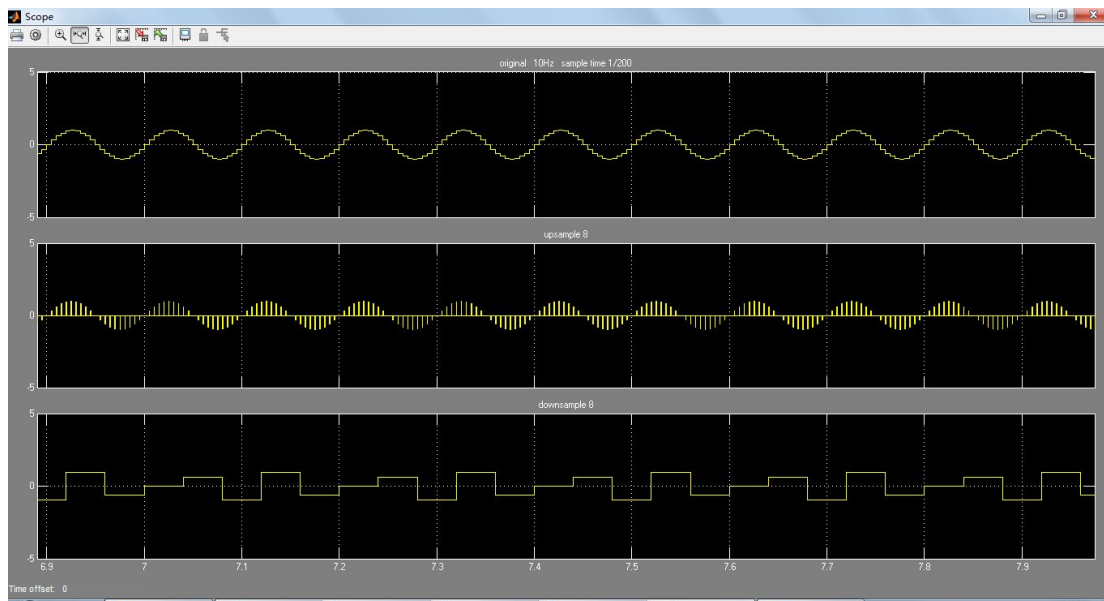
## Base-band modulation

### Why we have to up-sample ?

As we all know, Nyquist theory requires the sampling rate to be higher than double the highest frequency of the data signal. Even though the original signal meet the requirement of Nyquist theory, however, when base-band signal turn to the higher frequency band signal, the new frequency may be not meet the requirement any more. Therefore, the sampling rate should be raised before band modulation. This is realized by “inserting zeros” (i.e. Up-sampling).

How up-sampling and down-sampling affect the signals are shown as follows:

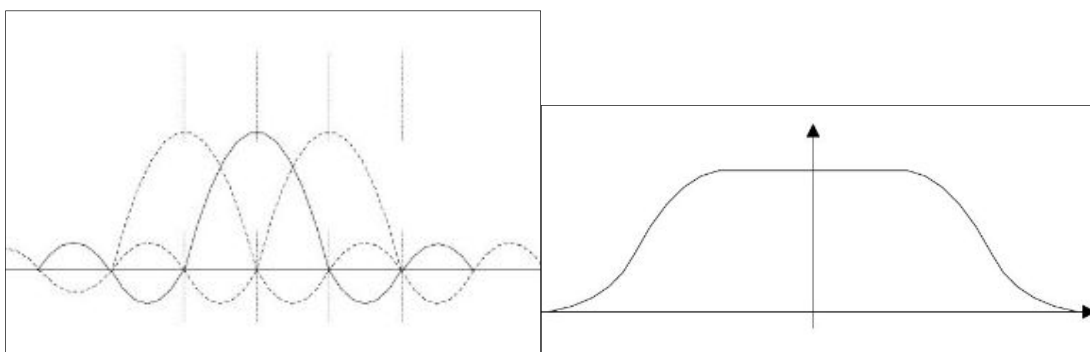
Figure: up-sampling and down-sampling (three sub-figures are original signal, 8 multiples up-sampling and 8 multiples down-sampling respectively).



Seen from the figure above, up-sampling is actually “inserting zeros”. Add  $(n-1)$  zeros ( in this system we have  $n=8$ ) after each sample point; down-sampling is actually “extracting”. Zero order holding on the sample point, lasting for  $n$  sample times. Up-sampling raise the sample rate to 8 times the original rate in digital domain. It does not affect the real frequency of signal in analog domain, the signal band-with is not raised either. Similarly, down-sampling is to extract one from every eight sample points, reducing the sampling rate.

Because the square wave has infinite band width in frequency domain, it is not transferable in channel. Therefore, it is necessary to reshape the wave after sampling. Various wave shapes are applicable for base-band signals. Raise cosine filtering ( square-root raise cosine filtering at both transmitter and receiver respectively). Adjacent symbols could compensate each other in frequency domain to get a flat spectrum. In time domain, the moment a given symbol reaches its peak, the other symbols are exactly zero. Since the digital communication system only cares about the value on the sample points, we just have to guarantee that there is no interference on sample points.

Figure: signal without ISI , in time domain (left), in frequency domain (right)



The multiple of down-sampling in receiver is not necessarily be consistent with the multiple of up-sampling in transmitter. There inserts some zeros in up-sampling. After filtering, values on those points turn to (or approximate to) the actual value of the signal, having the same effect as “inserting” (different from inserting zero). Afterwards, when extracting values, every point, whether the original points or the ones inserted later, are all eligible for extracting. Therefore it is okay if the multiple of up-sampling and down-sampling are different.

The sample rate of the carrier is 8 times of the frequency of signal. That is, there are 8 sample point in each cycle of carrier. The offset of sample points ranges from 0 to 7, 8 possible offsets in total. In order to figure out the best sample point, we adopt a method of trying them one by one.

First, separate the signal into 8 branches, setting the offset of down-sampling to be 0 to 7 respectively. Then, get the sum of absolute values of the sampled values in each branch. Add up , calculate the average of each, then display in the display block. The result of average value with different offsets are different. The branches sampling at exact peak and valley of the wave will display the largest value.

We can tell the noise and interference of the signal from the eye diagram. The bigger eye is, the higher quality of the signal. The mechanism of eye diagram is to capture the wave of signal every half cycle, then overlap several figures together. If the value of wave is far from the central value ( like zero in this case), that means the signal is of high quality, clear enough to make a decision. The decision is supposed to be made at the central point of the eye, namely at the best sample point.

#### Band modulation ( demodulation at receiver )

The ultimate goal of a communication system is to convey message from one side to another side. The very essence of message is the information in a certain pattern, namely a certain regulation. The task of the transmitter is to load the pattern or regulation on to a physical object which could travel for a very long distance. Our ancestors discovered the wave to have those qualities. So we assign the “regulation” to the wave. As for wireless communication, wave travels via air; as for wire communication, wave travels via waveguides.

Why we need modulation ? That is because different waves has varies properties and qualities while traveling, longer waves are easier to scatter than shorter waves. Therefore, we prefer higher frequency waves to act as our media. Apart from that, since there are many users sharing the same wave spectrum. A system is supposed to occupy the least source of band width. Band width is divided and allocated by International organization for standardization. When designing a system, one should modulate the signal to the exact frequency allocated to you. In this project, we

1.25 MHz as central frequency.

What the modulation does is to let the high-frequency wave embody the “regulation” we want to convey. How could the high-frequency wave carry the information of “regulation”? Let's have a look at the properties of wave. It has amplitude, frequency, phase. We can alter those three or their combination to make the wave change according to the “regulation”.

The high-frequency wave (i.e. carrier  $c(t)$ ) is sine wave, which has a single frequency. The three major modulations are amplitude modulation, frequency modulation and phase modulation. The one which already carried the “regulation” is called modulated signal,  $m(t)$ . After modulation, the signal is more suitable for transmission, thus reducing the distortion of signal. Moreover, it is a good practice for a lot of users to share the frequency spectrum.

In simulink, we use sine block in discrete signal box. What we try to do is to simulate the analog sine modulation with fix-point discrete computer program. Therefore, frequency of the sine wave must be much higher than sample rate (i.e. symbol rate)

In our system, after base-band modulation and spread spectrum, data rate has become  $2048*64$  bps, the frequency of the sine wave to be modulated is going to be  $2048*64*100$  bps, furthermore, sample rate of the sine wave is supposed to be  $2048*64*100*8=131$  M bps

In the practical transmitting and receiving, the receiver does not know about the information about of carrier. However, we need the original carrier to demodulation on the receiver. So it is necessary to restore the carrier. We adopt square loop carrier restoration method.

Follow the principle of the equation below, we first square the BPSK signal out from AWGN channel, take the high frequency part, which is twice the frequency of carrier. Phase-locked-loop is used to separate this carrier.

$$e(t) = m^2(t) \cos^2 \omega_c t = \frac{m^2(t)}{2} + \frac{1}{2} m^2(t) \cos 2\omega_c t$$

Divide-2-frequency is a bottleneck. We come up with a method to manage it with counter. Every time when the signal cross zero, the counter plus one, every 2 times to generate a “hit”, namely output. See the figure below, it turns to periodical square wave. What we want, is a sine wave with the same phase and amplitude as the carrier on transmitter. Therefore, feed it in a low-pass filter, then we get the original carrier.

Set sample frequency ( $F_s$ ) at 1000 Hz, pass-band frequency at 100 Hz, stop-band frequency at

105 Hz. Original carrier could be restored well. (see figure below)

Figure: square loop carrier restore ( The three sub-figures are original carrier, half-divided frequency carrier and carrier after low pass filter respectively.)

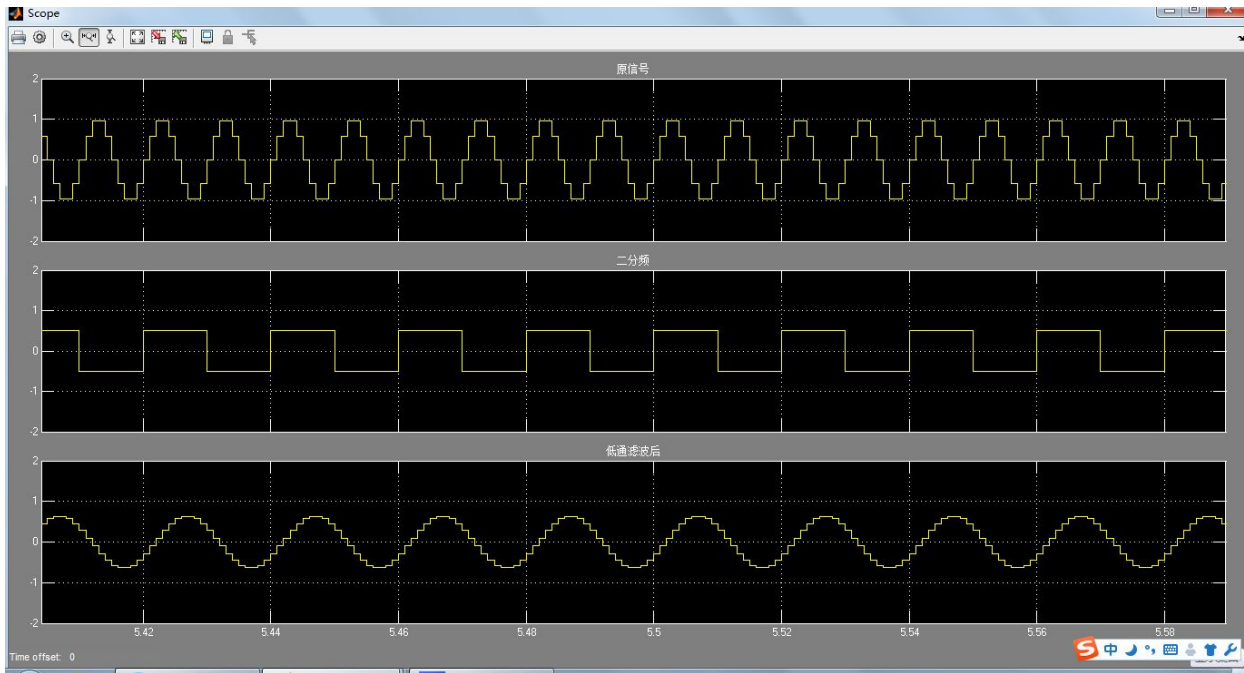
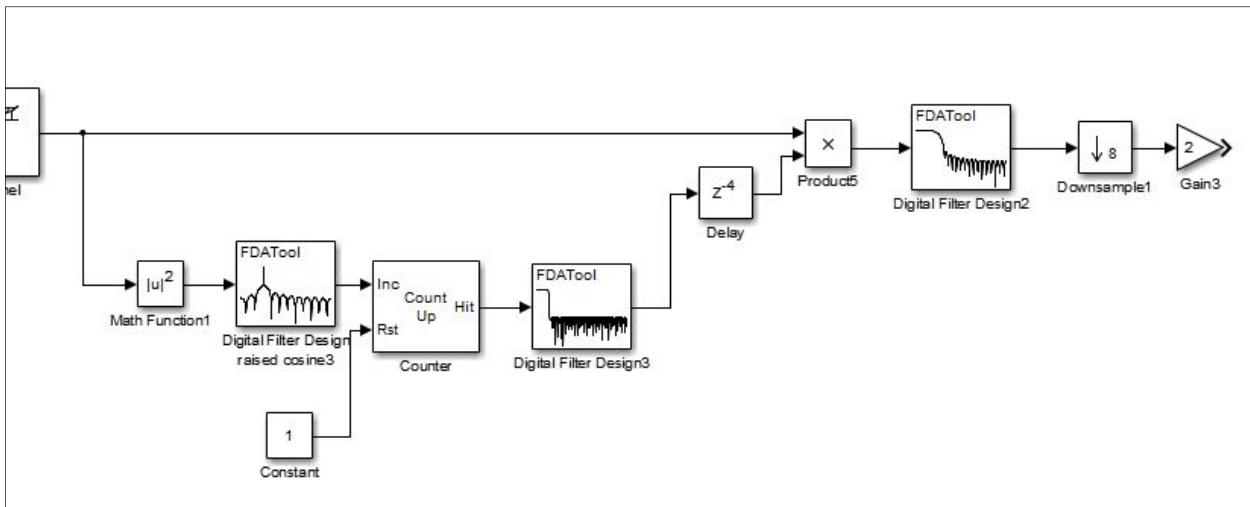


Figure: block diagram of square loop carrier restore in the system



Square loop carrier restore is applicable to situations which has a high SNR (signal to noise ratio). When SNR is relatively low, it does not work well. Therefore, we get the frequency and phase of carrier with the aid of pilot ( details will be elaborated later), instead of the square loop carrier restore method.

After the carrier restore, we just restore the frequency, but not the phase. It is necessary to down-sample before restoring phase. Because down-sampling is able to reduce the data rate, making it easier and more accurate to restore the phase. Moreover, if sampled at the best sample



point, with larger difference in values, it lowers the possibility of wrong decision.

### BPSK modulation

There are many kinds of digital band modulation, FSK, PSK, ASK, etc. This system adopts BPSK modulation (binary phase shift key modulation). When viewing the diagram, our system is kind of similar with QPSK modulation. I am going to show the difference between what we have done and OPSK modulation, by introducing the characteristic of both.

QPSK transmit 2 bits of information each time after modulation. Those information, or “regulation” as we mentioned above, is conveyed through four different phases of the carrier, namely 0,  $\pi/4$ ,  $\pi/2$ ,  $3\pi/4$ . The process is first convert the input binary series into  $m$  parallel branches,  $m = \log_2 M$ . Data rate of each single branch is  $R/m$ , in which  $R$  is the data rate of the input serial sequence,  $m$  is the number of branches.

I/Q signal generator is to convert every  $m$  bits to a pair ( $I_n$ ,  $Q_n$ ), two branches of sequence, with each of which modulated by cos and sin signal. Then add them up to get a QPSK signal in the end.

BPSK load the information onto one-dimension space, using only one PN sequence to spread the spectrum. QPSK, however, load the information onto two-dimension space, applying PNI and PNQ two different PN sequences to spread the spectrum.

Channel utilization of QPSK is twice that of BPSK, but the error rate is higher as well. QPSK need higher SNR (signal to noise ratio) than BPSK to meet the same requirement of BER (bit error rate).

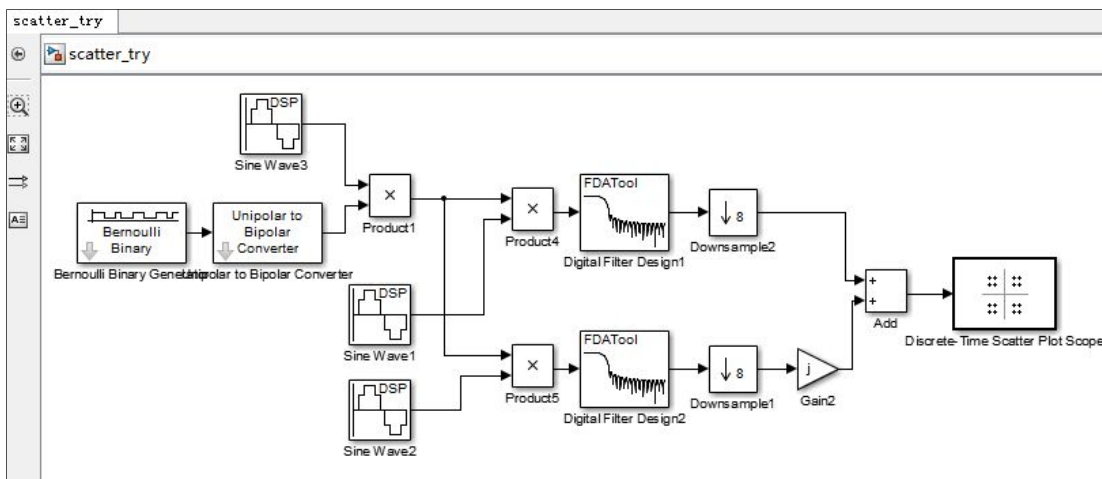
Our power line carrier communication system works in a low SNR environment, so the primary concern is reliability, which allows the system to be resistant to high interference. Weighing various factors we choose BPSK modulation. It seems that our system also have sin and cos, two orthogonal branches. But that is not QPSK actually. What we have done is separate the signal into two orthogonal signals to modulate.

Orthogonal demodulation is conducted with I, Q, two channels. In ideal situation, BPSK only results in 0 or  $\pi$ , two phases. There are only two points on real axis. In real situation, however, there might be phase shift during transmission. Points may rotate on constellation graph. components fall both on the real axis, but also on imaginary axis. (assuming we describe the orthogonal I, Q signals in complex plane).

For I branch of signal, namely cos signal, after demodulation, we get real part of the original signal; For Q branch of signal, namely sin signal, after demodulation, we get imaginary part of the original signal. Feed the two parts into the block, we get the constellation graph of receiver.

After Signals multiplied by the cos or sin wave, they are supposed to go through low-pass filter, to get rid of the high frequency part. Square-root raised cosine filter hold a concurrent post of the function of low-pass filter. Therefore, placing a square-root raised cosine filter here is enough. Additionally, a gain block of twice is applied here, allowing the receiving signal to have the same amplitude as transmitting signal theoretically

Figure : in-phase channel and quadrature channel demodulation block diagram in Simulink (it is for partial test, different from the one incorporated in the final system)



phase correction


Constellation diagram and phase offset

Why we separate one signal into two in the part we discussed above ? Because the two branches of signals could be applied to the two coordinates of constellation diagram, so that we get a constellation diagram indicating the phase offset of the system. In the final system, there is no need for two branches, only one is supposed to remain. In simulation, we can figure out the phase of the local carrier, then trying to keep the same pace with the transmitting side.

Next step is phase correction. It is actually a process of compensation. First calculate the phase of the current constellation point, getting the phase shift during the transmission. Then compensate the phase shift to restore the correct phase, rotating the constellation points back onto the real axis, namely 0 and  $\pi$ .

phase angle could be calculated by function  $\arctan(Q/I)$ . Definition domain of  $\arctan$  lies in region  $(-\pi/2, \pi/2)$ . If the constellation point locates at the first or third quadrant, the phase process will rotate them onto  $-\pi/2$  and  $\pi/2$  on imaginary axis. That is a mistake. Therefore, all the phase angle in those quadrants should be modified first before the phase correction.

s-function of phase correction is as follows:



```

1  function [sys,x0,str,ts]=phrase_revise(t,x,u,flag) % 相角校正
16 function [sys,x0,str,ts]=mdlInitializeSizes
17     sizes = simsizes;
18     sizes.NumContStates = 0;
19     sizes.NumDiscStates = 1;
20     sizes.NumOutputs = 1;
21     sizes.NumInputs = 2;
22     sizes.DirFeedthrough = 1;
23     sizes.NumSampleTimes = 1;
24     sys = simsizes(sizes);
25     x0=[0];
26     str=[];
27     ts=[1/131072 0]; %继承
28 function sys=mdlUpdate(t,x,u)
29     if((u(1)>=0)&&(u(2)>=0))
30         sys(1)=u(1);
31     end;
32     if((u(1)>=0)&&(u(2)<=0))
33         sys(1)=u(1)+pi;
34     end;
35     if((u(1)<=0)&&(u(2)>=0))
36         sys(1)=u(1)+pi;
37     end;
38     if((u(1)<=0)&&(u(2)<=0))
39         sys(1)=u(1);
40     end;
41 function sys=mdlOutputs(t,x,u)
42     sys(1)=x(1);

```

### Acquisition and synchronization

After receiving the signal, the receiver first figure out the precise phase of PN by acquisition. Generate a local PN sequence with exactly the same phase with that of the received PN sequence. Then it can restore the original signal.

De-spread spectrum is typically done by the correlator. Matched filter is an alternative, but it is much more complicated and not necessary in most cases. Therefore, we employ correlator in

system design.

Sliding correlator processes the two PN sequences bit by bit, moving one bit at a given time. The searching efficiency is relatively low. There are two solutions: (1) add some more correlators to do the work in parallel. (2) limit the searching scope to a certain range, instead of searching along the whole sequence. Our system has already met the specification requirements even without applying two solutions. However, those two points are brilliant for the system to improve its performance.

synchronization generally fall in two categories, rough synchronization and fine synchronization.

(1) rough synchronization: The receiver do not know about whether the transmitter is sending any data at the moment. Rough synchronization is to search and capture the signal in a given range. The goal of rough synchronization is to reduce and control the phase difference within a symbol.

(2) fine synchronization (tracking): receiver constantly adjust the phase of local PN sequence through early-late-gate, keeping it synchronous with received signal dynamically. Even if encountered with exterior disturbance, a feed back mechanism will resume the synchronization.

The key problem for acquisition is to figure out the delay during transmission, so that we can find out the phase difference. We find it hard to solve the problem directly, since there are numerous factors on the link which could cause the delay. So we turn to another indirect way to find the delay.

As we know walsh has the property of orthogonality. When two identical ( both 64 in length and bear the same index ) and aligned walsh sequences multiply, (after 0,1 has turned into 1,-1), product of each bit is 1 in a row. Adding up the products of correlation range, the sum of products may reach 64 theoretically. Seen on the figure, there exists a correlation peak. In this way, we are possibly readily find the phase difference.

Since we choose the 64 order walsh code, the symbol rate is supposed to be  $1/64$  of the rate of that of the walsh code. ( or  $1/128$ ,  $1/192$ ....). after the multiplication of walsh and data, each bits has a 1, therefore the correlation is formed.

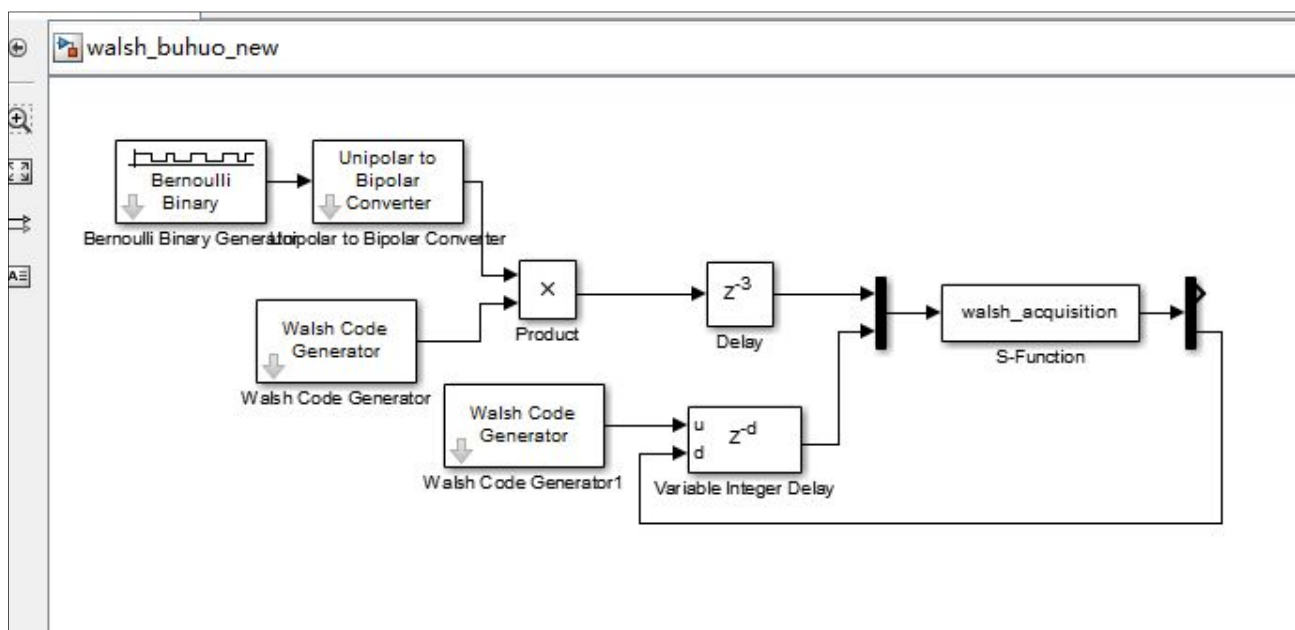
As long as the requirement above is satisfied, we do not care the difference between symbol rate of signal and walsh. Height of correlation peak is determined by length of correlation range. In theory, the two indicators should be the same. In practical, there could be a deviation less than a symbol width. Like  $1/6$  or  $1/5$  symbol width. Although not aligned exactly, under this judging mechanism, it has to be determined to be aligned. Therefore, correlation peak may be lower than

theoretical height.

The longer of the correlation range, the more visible correlation peak is. The cost, however, is the reduction of efficiency, since it requires more multiplications and additions. We set constraint length to be 192 ( i.e. Calculate sum of products every 192 bits) in the system. During the simulation, for the sake of better visual effect, we set the constraint length to be 64. We use 64 order walsh.

Some tips on how to Write s-function effectively:

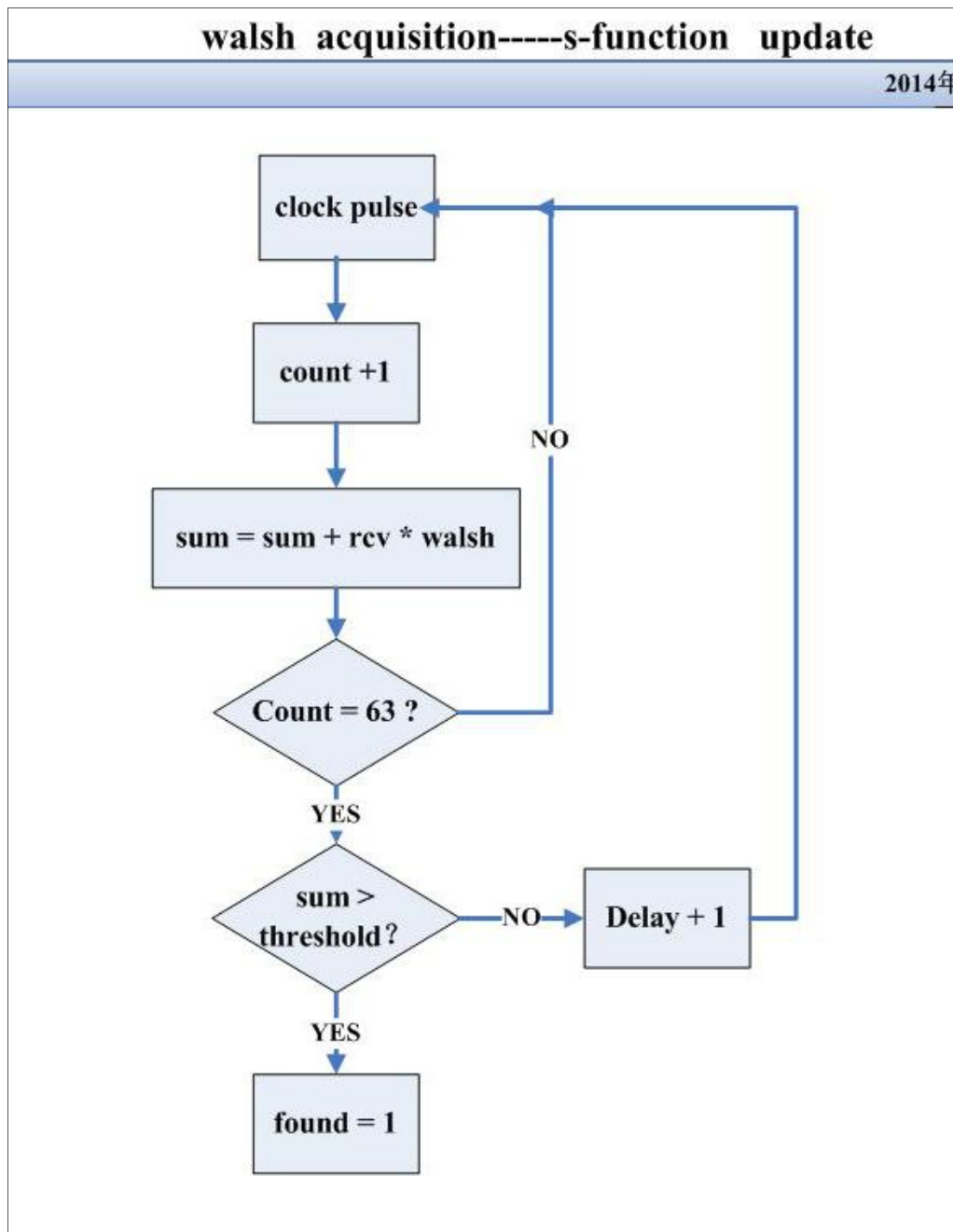
Determining the structure of the whole system first. Then figure out the exact position “s-function” block is located in the system, deciding its function.



Determine the input, state variable and output of the s-function

Input “u”	State variable “x”	Output “y”
Received signal “rcv”	Accumulation “sum”	Delay “delay”
Local walsh code “walsh”	Sign showing whether it has already found correlation peak successfully “found”	
	Count variable “count” (ranging from 1 to the correlation length)	

Program diagram in s-function ‘s update part:



PN and walsh have similar functions in some aspects; they have their own strength and weakness in others. In wireless communication standard IS-95, short PN code, with a cycle of  $2^{15}-1$ , is to identify different base stations; 64-order-walsh is to identify different channels; long PN code, with a cycle of  $2^{42}-1$ , is to identify different users. PN code has more applicable offsets, which increase the amount of effective PN sequences. Walsh code has a better orthogonality. In our system, two codes are combined to improve the performance.

s-function for acquisition of PN and walsh are as follows:



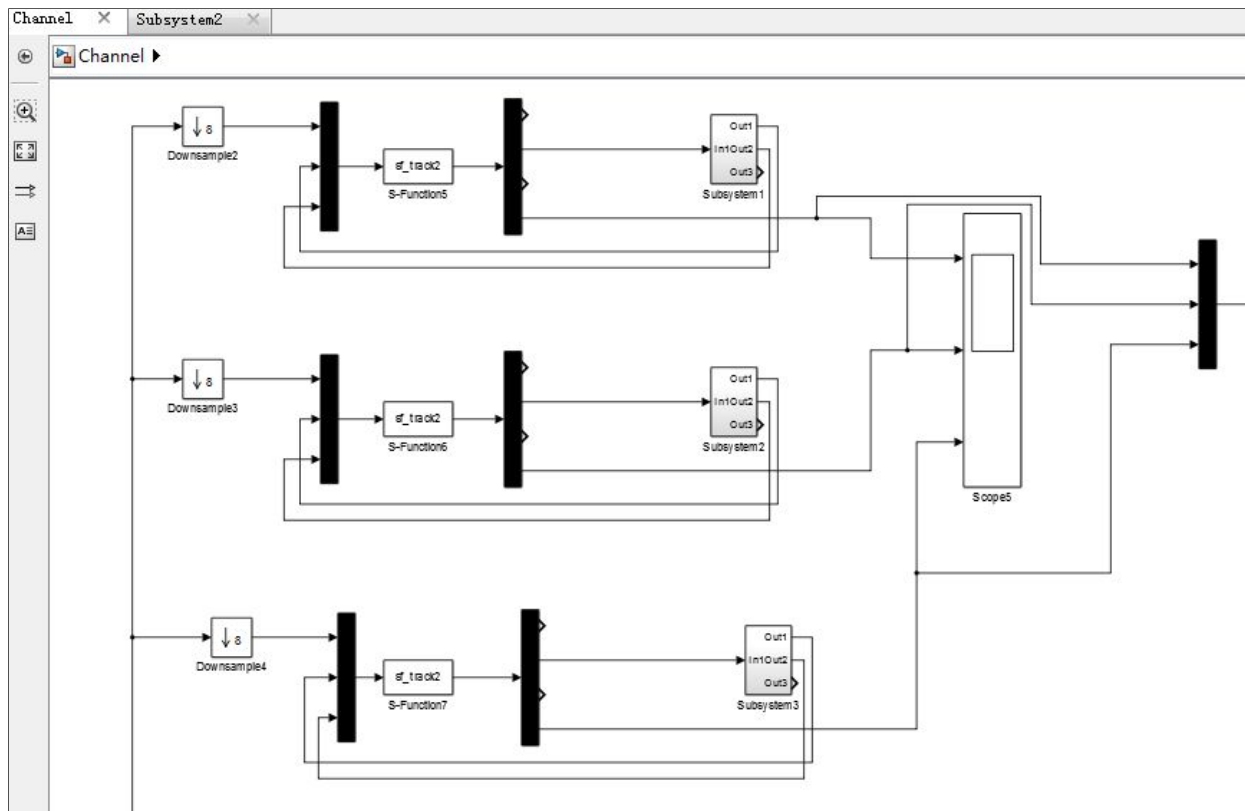
```
correlat... x
1  + function[sys,x0,str,ts]=correlation(t,x,u,flag) % 做64长度的相关
18 - function [sys,x0,str,ts]=mdlInitializeSizes
19 -     sizes = simsizes;
20 -     sizes.NumContStates = 0;
21 -     sizes.NumDiscStates = 4;
22 -     sizes.NumOutputs = 1;
23 -     sizes.NumInputs = 4;
24 -     sizes.DirFeedthrough = 1;
25 -     sizes.NumSampleTimes = 1;
26
27 -     sys = simsizes(sizes);
28 -     x0=[0,0,0,0];
29 -     str=[];
30 -     ts=[1/131072 0];
31
32 - function sys=mdlUpdate(t,x,u)
33
34 -     count=x(1);
35 -     sum_out=x(2);
36 -     sum_hold=x(4);
37
38 -     if(u(1)>10000)
39 -         en=u(1);
40 -     else
41 -         en=x(3);
42 -     end;
```

```

44 - if(en>10000)
45 -     if(count<63)    %计数 31循环
46 -         count=count+1;
47 -         sum_out=sum_out+u(2)*u(3)*u(4);
48 -         sum_hold=sum_hold;
49 -     else
50 -         count=0;
51 -         sum_hold=sum_out;
52 -         sum_out=0;
53 -     end;
54 - else
55 -     count=0;
56 -     sum_out=0;
57 -     sum_hold=0;
58 - end;
59 - sys(1)=count;
60 - sys(2)=sum_out;
61 - sys(3)=en;
62 - sys(4)=sum_hold;
63 - function sys=mdlOutputs(t,x,u)
64 -     count=x(1);
65 -     sum_out=x(2);
66 -     sum_hold=x(4);
67 -     if(count==63)
68 -         sys(1)=sum_out;
69 -     else
70 -         sys(1)=sum_hold;
71 -     end;

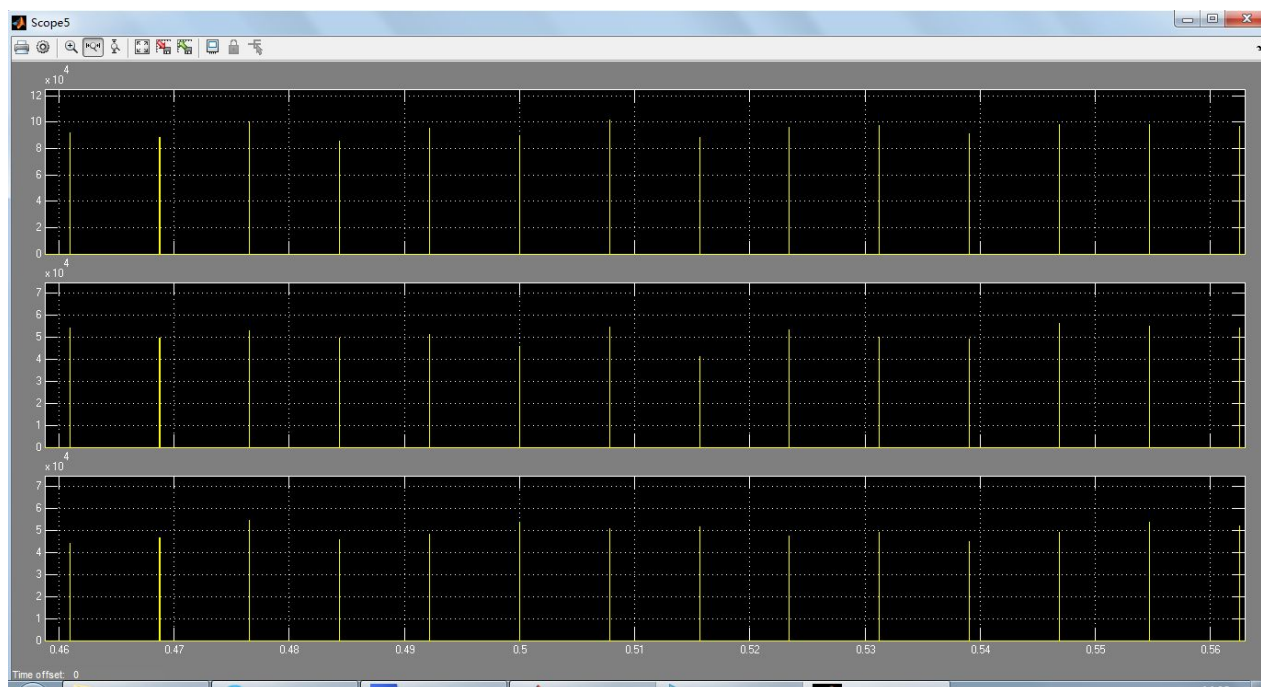
```

PN, walsh1, walsh2 are synchronized in three separate correlators. (see figure below)  
figure : block diagram of PN, walsh1, walsh2 correlator in simulink



Outputs of three correlators are shown in the figure below.

Figure: PN, walsh1, walsh2 acquisition output

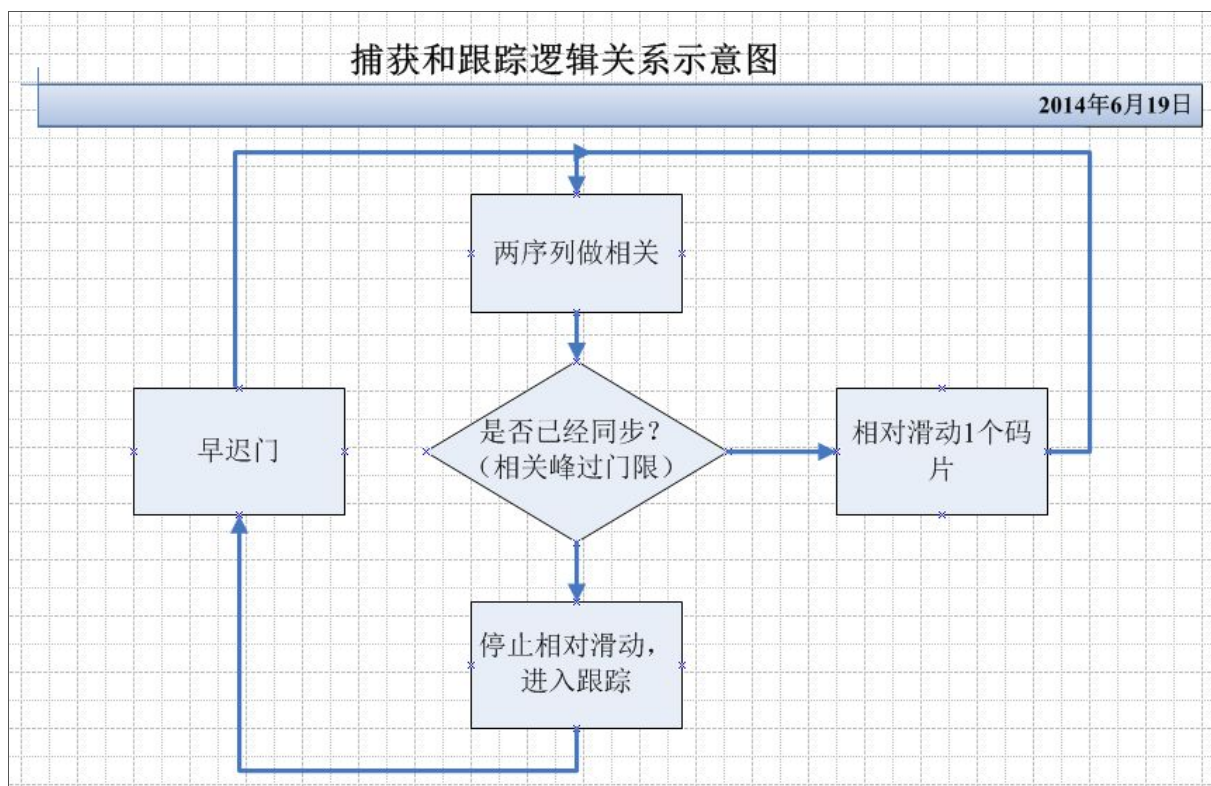


### Early-late-gate tracking

The sliding correlation introduced above is the rough synchronization. After the successful synchronization, the system comes to the fine synchronization, namely tracking the synchronous

signals in case some turbulence will damage the synchronization.

Early-late-gate is commonly used to realize this purpose. The schematic diagram showing logical relation of acquisition and tracking is shown below.



Received signal and one-symbol-advance-local PN sequence do correlation; see if the correlation peak exceed the threshold. If it does, we get result 1. Received signal one-symbol-delay-local PN sequence do correlation, if the outcome exceed the threshold, we get result 2. Subtract the two results, it can be determined whether to move forward or backward, according to the positive or negative of the subtraction.

one-symbol-advance-local PN and one-symbol-delay-local PN could be elicited by the two adjacent shift register of PN generators respectively. Correlation can be realized by multiplying unit, band-pass filter and square-law envelope detector in combination.

Figure (below) : logic block diagram of early-late-gate

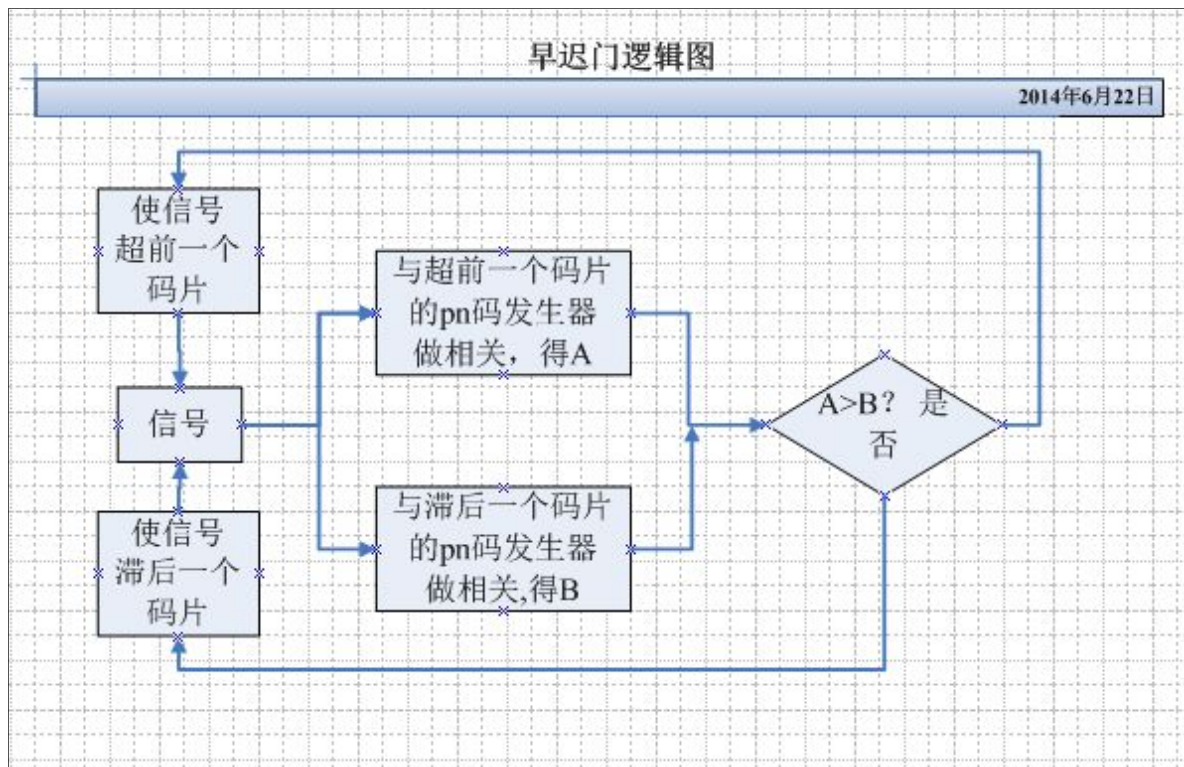


Figure (below): early-late-gate in the system

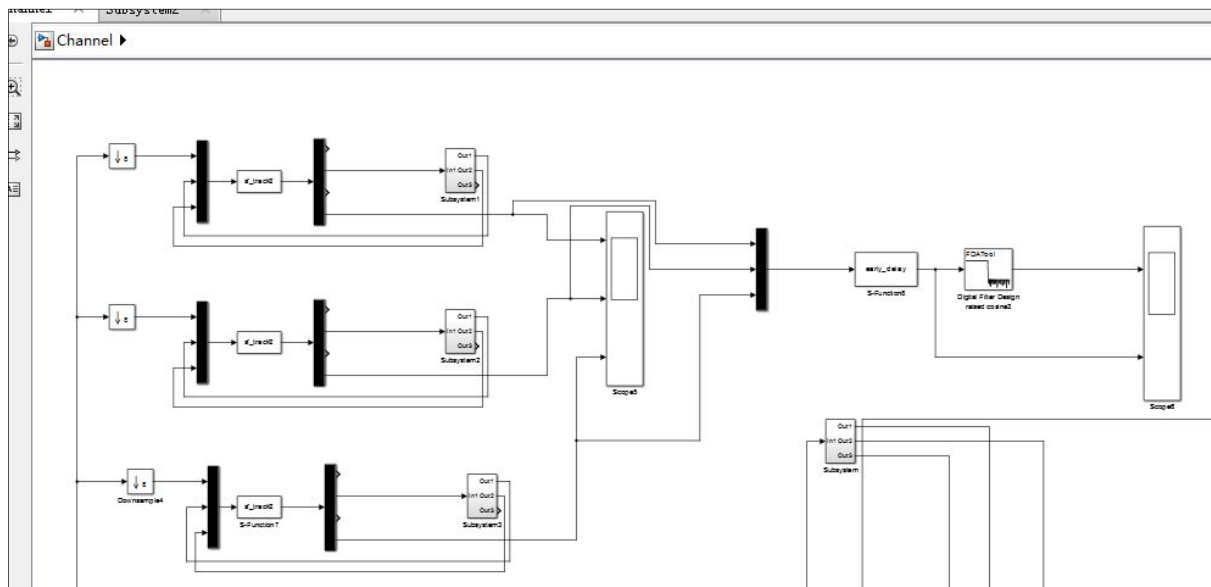
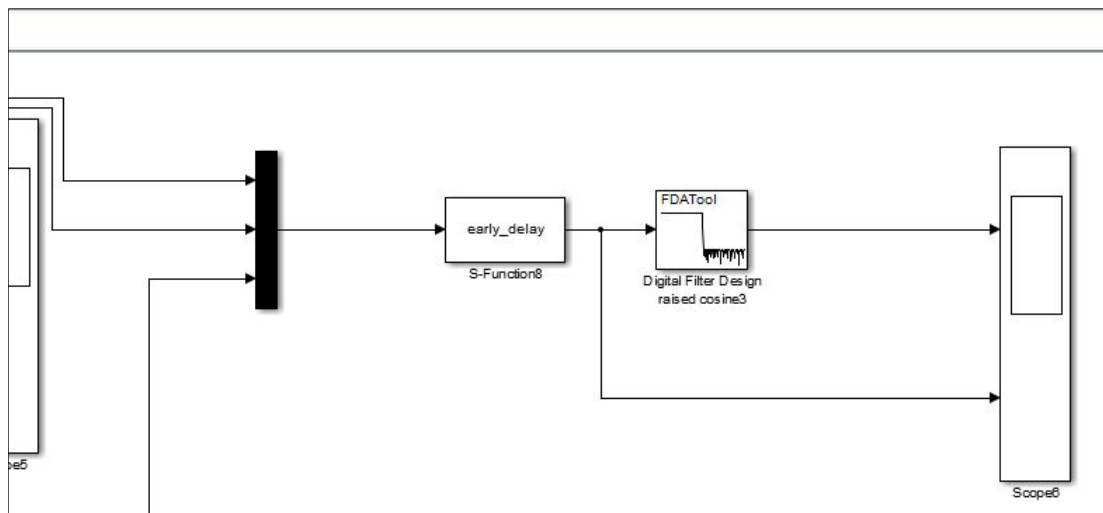


Figure (below): simulink block of early-late-gate





s-function of early-late-gate:

```

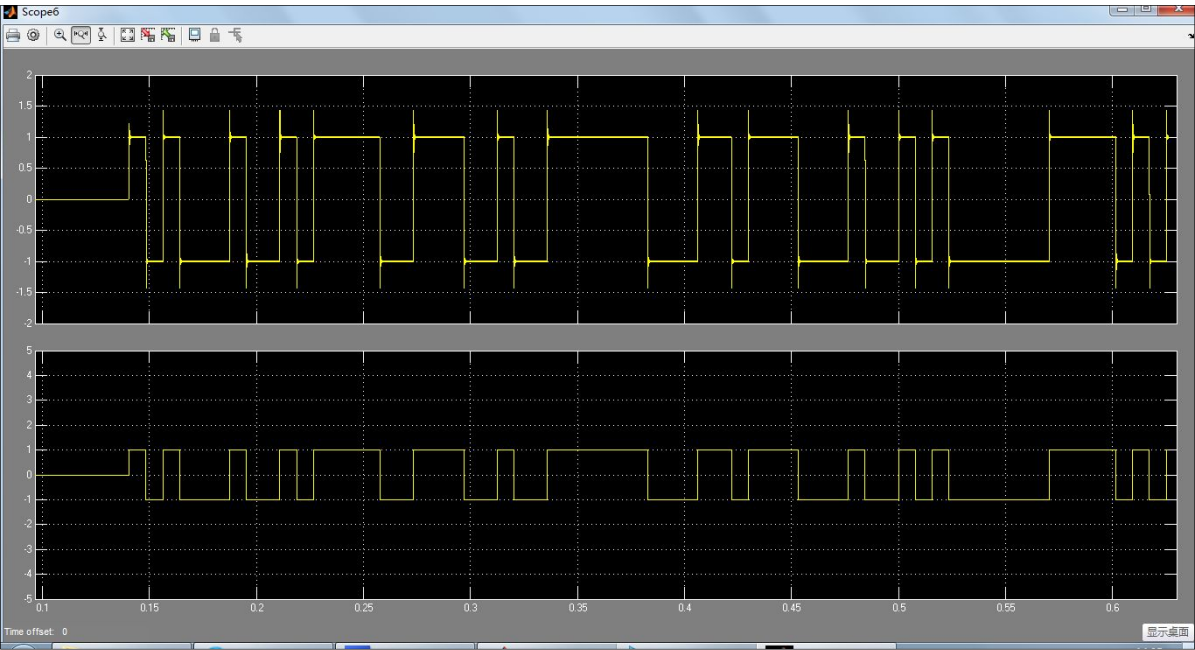
1  + function[sys,x0,str,ts]=early_delay(t,x,u,flag) % 早迟门
18
19  + function [sys,x0,str,ts]=mdlInitializeSizes
31
32  - function sys=mdlUpdate(t,x,u)
33  -     if(u(1)>60000) % 相关值超过门限
34  -         if(u(2)>u(3)) % early 比较大
35  -             sys(1)=-1;
36  -         end;
37  -         if(u(2)==u(3)) % 两者相同，说明刚好对齐
38  -             sys(1)=0;
39  -         end;
40  -         if(u(2)<u(3)) % late 比较大
41  -             sys(1)=1;
42  -         end;
43  -     else % 相关值没有超过门限
44  -         sys(1)=x(1);
45  -     end;
46
47  - function sys=mdlOutputs(t,x,u)
48  -     sys(1)=x(1);

```

Do subtraction of the two channels to judge the position change. See outcome below.

Figure : output of early-late-gate





Channel

One of the main objects of the design of communication system is to refrain interference, guaranteeing the reliability and effectiveness of the transmission.

There are many possible factors which may cause distortion in transmission. Generally, interference can be divided into two categories. adding interference and multiplicative interference.

Chart: categories of interference and corresponding simulation methods in this project

Adding interference: It comes from some unwanted signals added to the useful signal.		Considered (simulation method) or not considered in our system	Multiplicative interference: distortion caused by imperfect channel, which could not be represented simply as adding interference		Considered (simulation method) or not considered in our system
Rand om noise	Thermal Noise, cosmic noise or noise by electronic devices	Add “awgn channel” block, which is Gaussian distribution or narrow band pulse in time domain	Fading result from channel, distortion in amplitude and phase	Distributed everywhere in channel	Use “gain” block for amplitude distortion ; use “delay” block for phase distortion

Pulse interference	From atmosphere and industry		Frequency shift		The differentiation of phase shift is frequency shift
Sine interference	From adjacent channel or station, man-made interference	add a single frequency sine signal to the link	Phase shift		“delay” block
			Nonlinear distortion		Not consider

### Equalizer

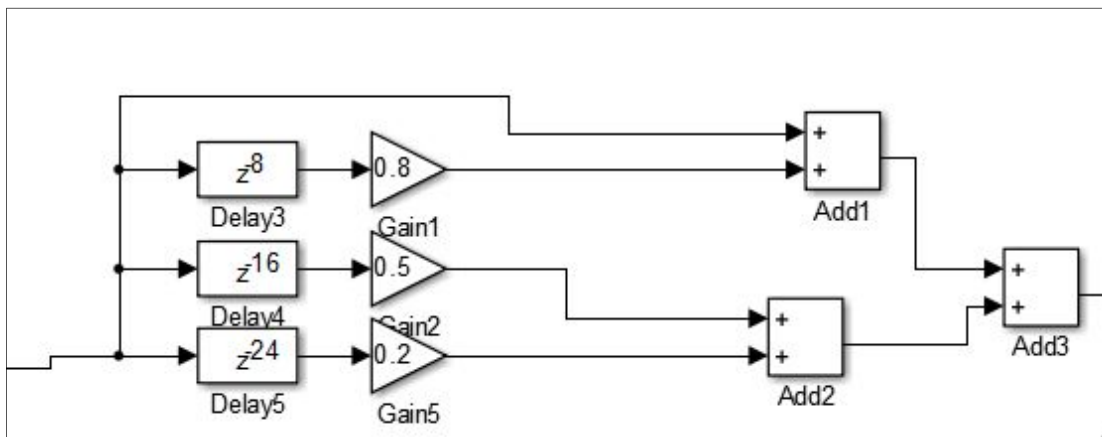
In order to compensate some channel deficiencies, some system employs equalizers. The main purpose for equalizer is to mitigate ISI in frequency selective fading channel. Another effective solution to release ISI is to spread spectrum, which we had already done in this project. Therefore, for simplicity, we do not add equalizer to the system.

Moreover, in this project, the symbol rate is 1.2288Mbps. Band width is relatively small; therefore frequency selective fading could be overlooked. There does not have to be a equalizer. However, if the frequency is as high as 10Mbps, equalizer is required.

### Multi-channel

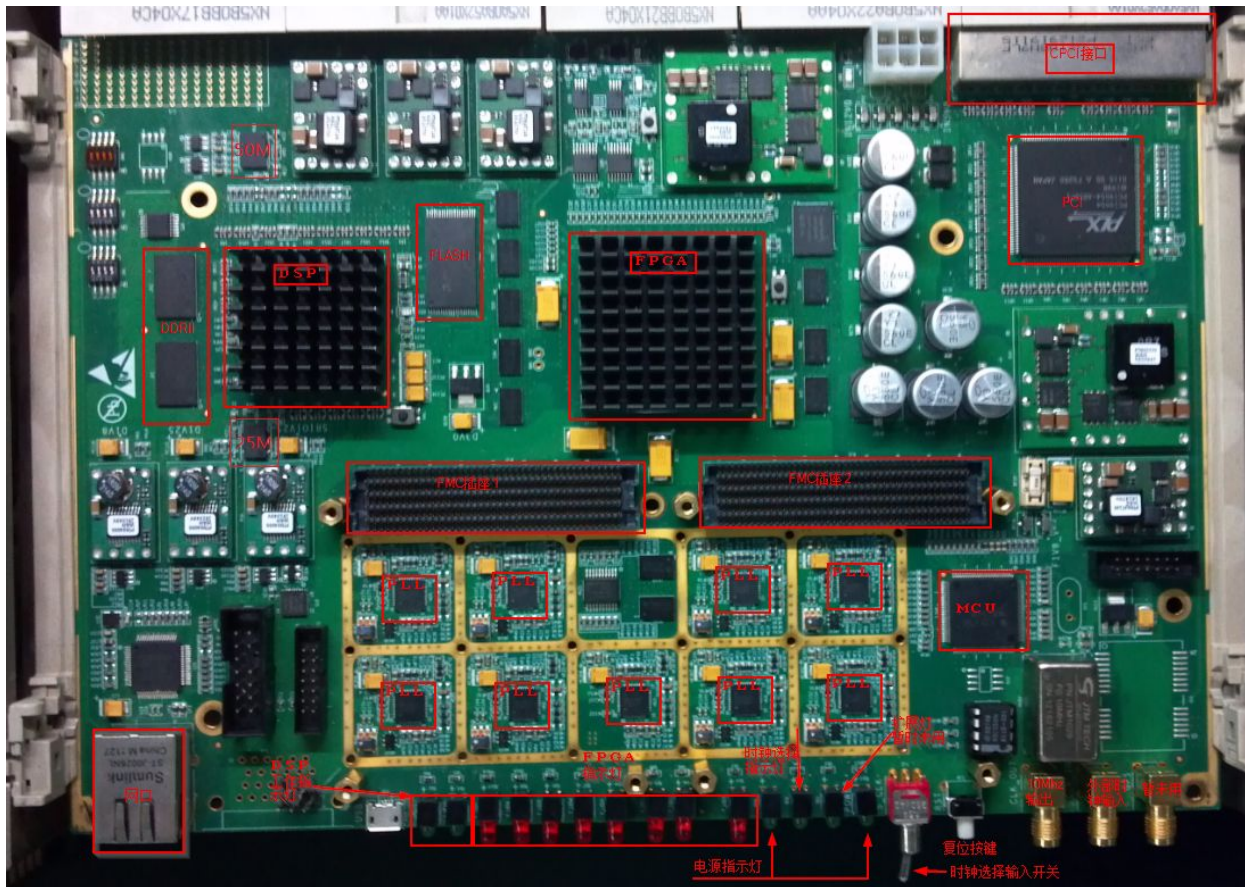
To deal with the problem of multipath fading during transmission, RAKE receiver is introduced. In simulation, we set four channels, applying different delays ( set 1, 0.8, 0.5 and 0.2 respectively) and different amplitudes. Then add them up.

Figure: block diagram for multi-channel in Simulink



After finishing the simulation in Matlab and Simulink of the whole system, it comes to the implement in FPGA and DSP.

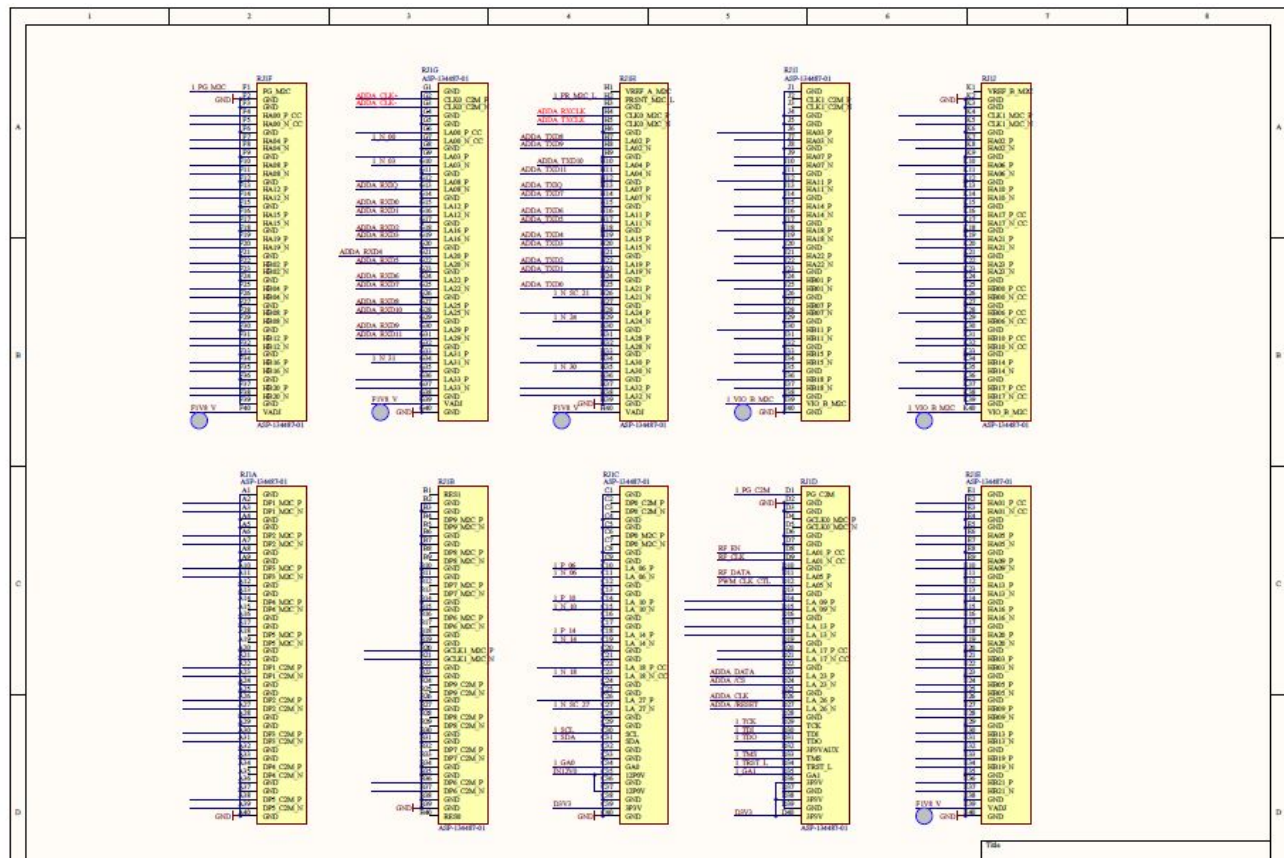
## PFGA and DSP integrated board



For DSP, we choose the chip TMS320C6455 from TI®. Frequency of it can be as high as 1GHz. The processing speed is 8000MIPS. It has a 64bit/133MHz EMIFA, which we use.

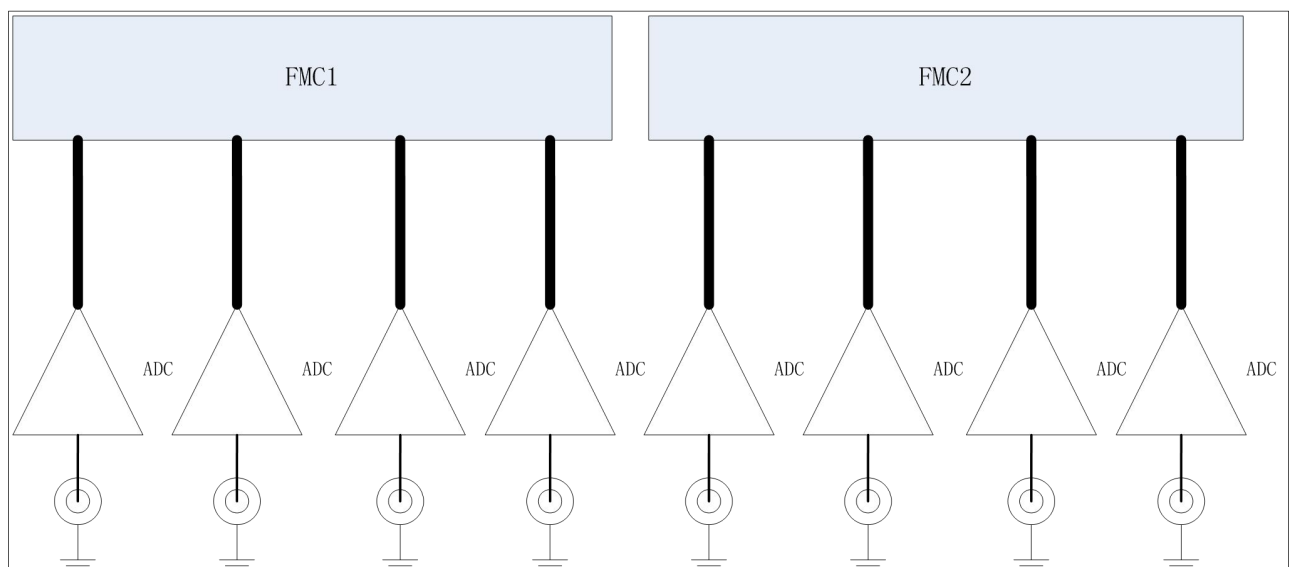
As for A/D and D/A convert, for better performance, we use an 8 channel ADC converter, which is not on the board. We use FMC( FPGA Mezzanine Card) to connect it to base-band board.

See interconnection of FPGA Mezzanine Card below.



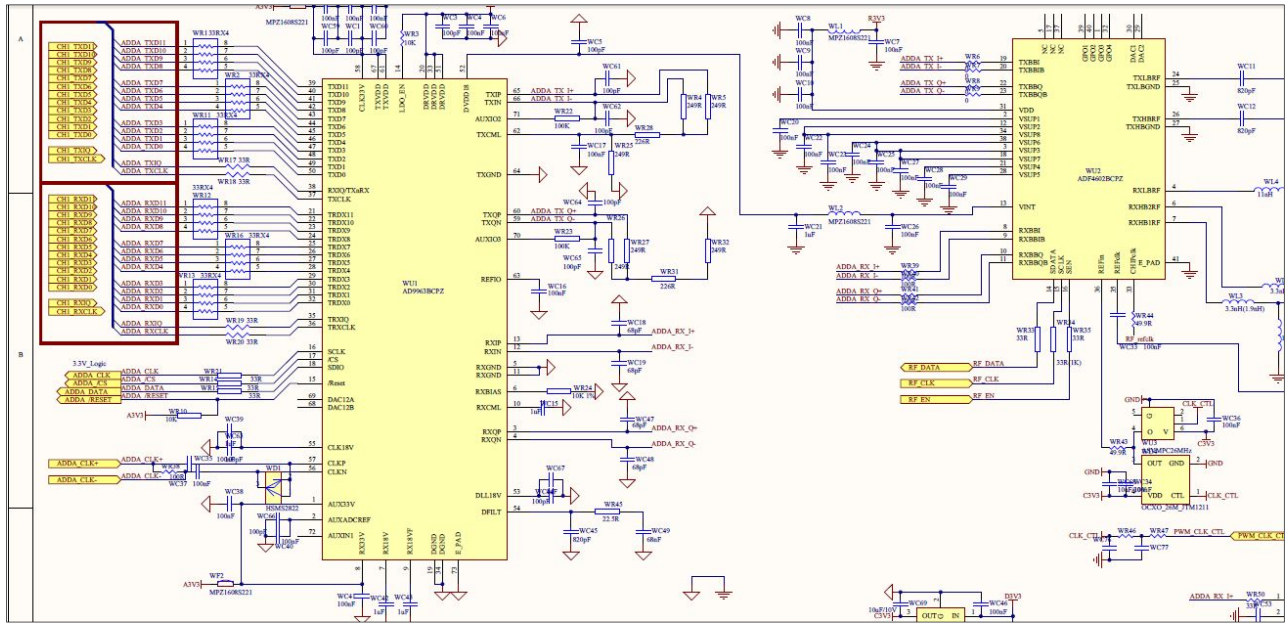
The overall structure of multichannel ADC transceiver is sketched below. There are 8 analog channels in total, each of which is independently controlled. For ADC chip, we use AD9963 ( 12 bit-10 bit-mixed-signal MxFE®) from ADI®.

Figure : schematic diagram for 8-channel-ADC transceiver



The integrated board provides 8 channels. In this system, we employ only one channel of them. See the graph below for the hardware connection of chip.





For PLL, we use chip LMX2541SQ2060E (Ultra-Low Noise PLLatinum Frequency Synthesizer with Integrated VCO) from TI®.

### EMIF interface

For DSP, we use chip TMS320C6455 (Fixed-Point Digital Signal Processor) from TI®.

The emif interface for TMS320C6455 is set as general memory interface, with four independent access spaces (CE2, CE3, CE4, CE5). Each memory space can be configured as 8 bit, 16 bit, 32 bit or 64 bit, synchronous or asynchronous mode.

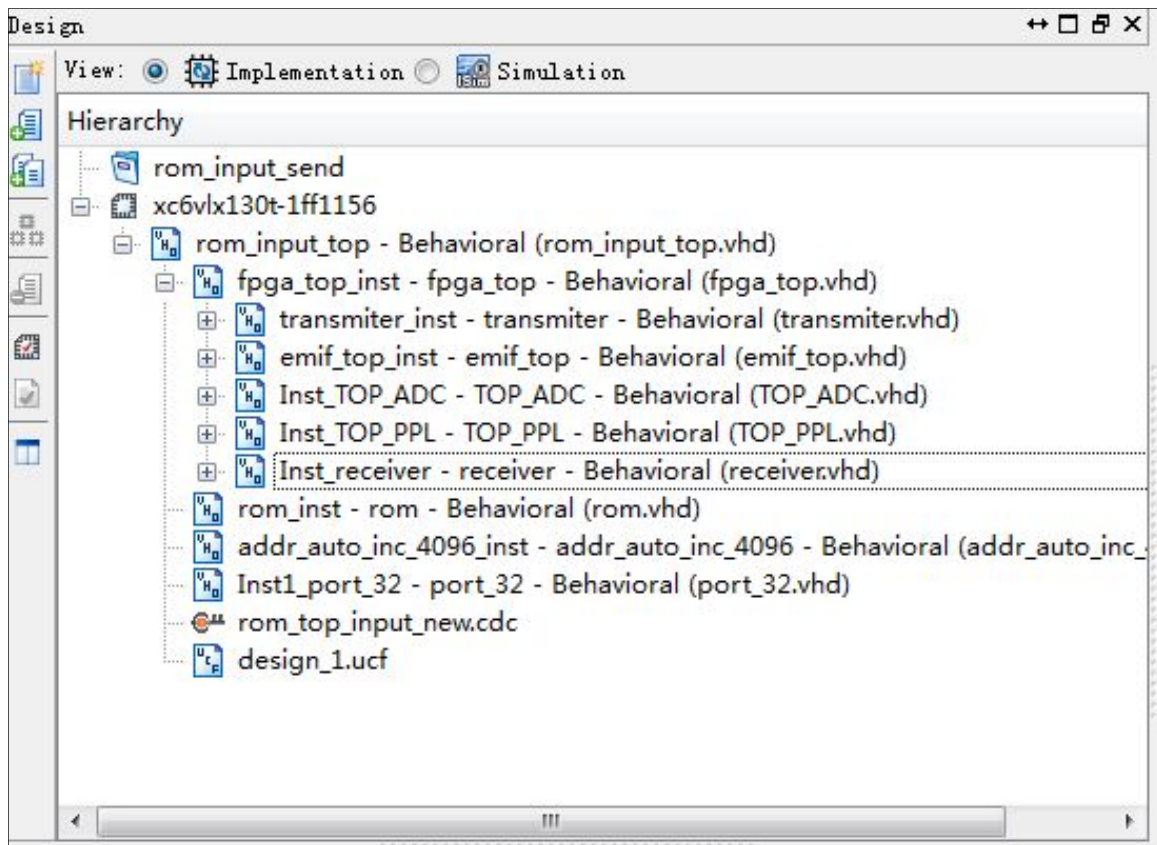
See detailed memory space and address in the char below.

EMIFA CE2 - SBSRAM/Async <sup>(1)</sup>	8M	A000 0000 - A07F FFFF
Reserved	256M - 8M	A080 0000 - AFFF FFFF
EMIFA CE3 - SBSRAM/Async <sup>(1)</sup>	8M	B000 0000 - B07F FFFF
Reserved	256M - 8M	B080 0000 - BFFF FFFF
EMIFA CE4 - SBSRAM/Async <sup>(1)</sup>	8M	C000 0000 - C07F FFFF
Reserved	256M - 8M	C080 0000 - CFFF FFFF
EMIFA CE5 - SBSRAM/Async <sup>(1)</sup>	8M	D000 0000 - D07F FFFF

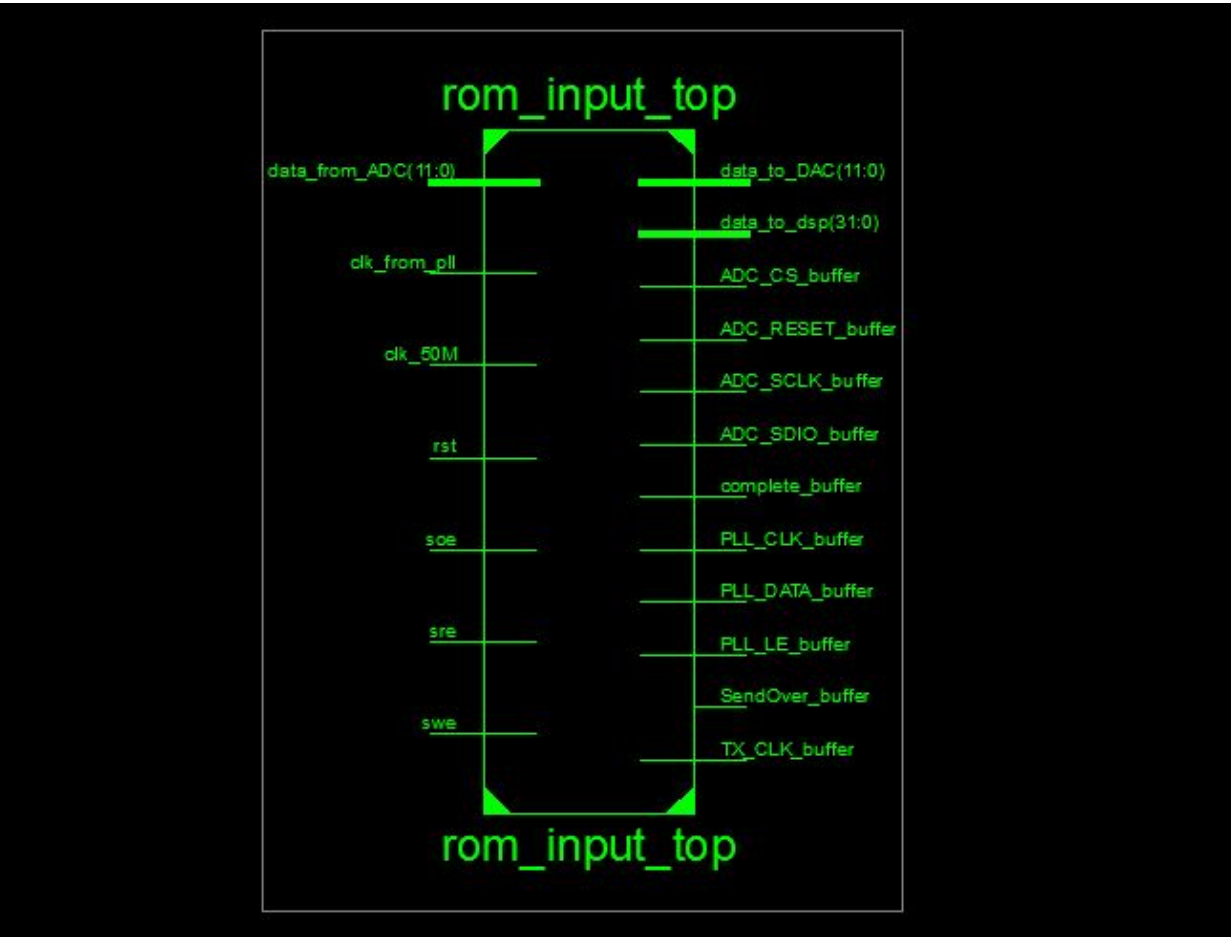
Our design use the EMIFA interface of DSP. When it comes to the hardware connection, DSP is connected to both FPGA and FLASH. The FLASH is for asynchronous mode. We adopt the synchronous mode so that we do not use FLASH.

Hierarchy design and concrete design for each block

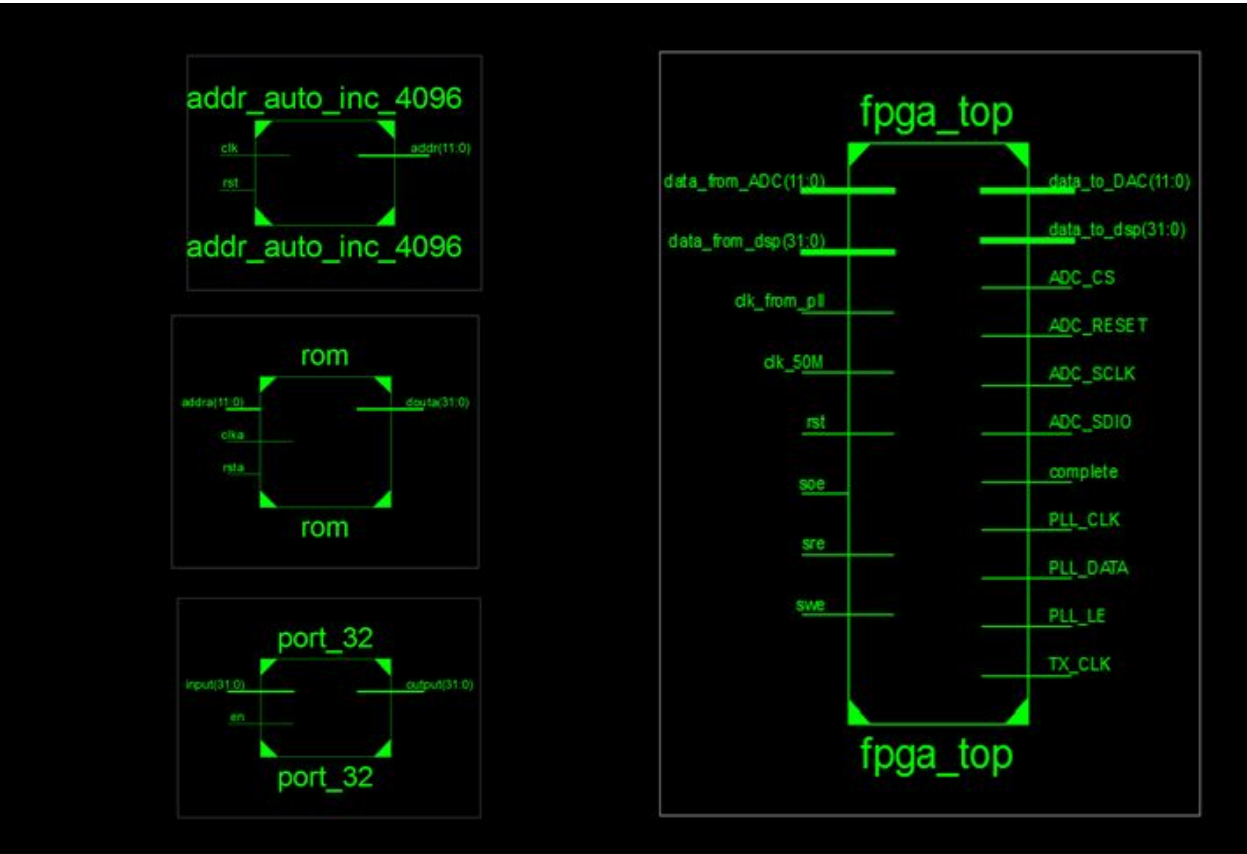
Hierarchy design in ISE is as follows:



“Rom\_input\_top” block is an additional cover wrapping the top block “fpga\_top”. The consideration for this design, is to let the top block carry a ROM of its own, which could automatically send data to it, making it easier to self-test. That is, it can verify its design without the coordination of DSP.



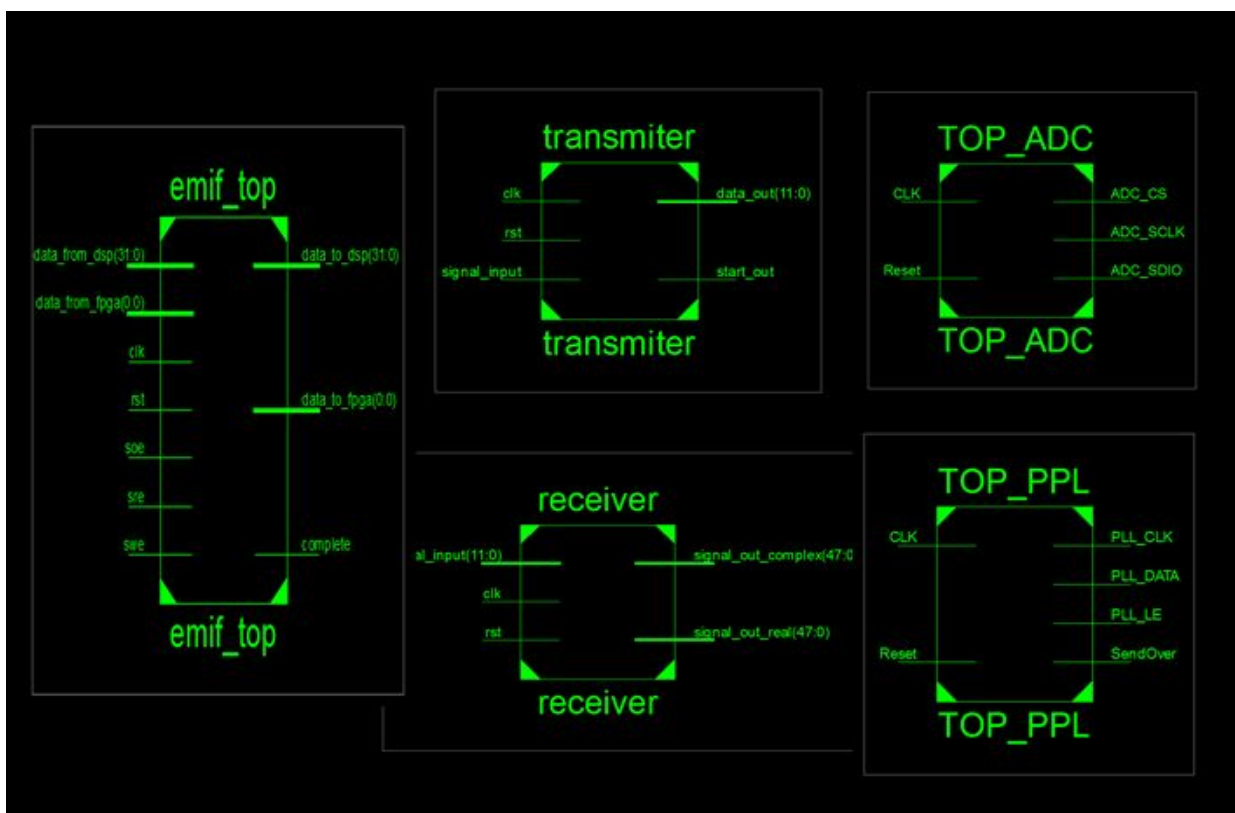
See interconnect of block “Rom\_input\_top” below.





The function of block “addr\_auto\_inc\_4096” is to automatically add one to the address with each clock pulse. Each time the rising edge of clock pulse comes, its output address plus one. In this way, we actually do not use the addressing function, since we do not need to address flexibly, just to read and store unit by unit could reduce its complexity. Its output address will be sent to block “rom”, fetching data in corresponding unit. Data stored in “rom” is 32 bits in length. Block “Port\_32” is a guarding logic. It has a input signal “en”; data reading and writing could process only when “en” is active.

“pfga\_top” is the ultimate top block for overall functions. See its interconnections below.



Block “emif\_top” serves as the connection of FPGA with emif com in DSP. It contains a ping-pong RAM, which can temporarily store data sent from DSP. We need ping-pong RAM because the speed of data transfer via emif is much fast than that of data procession in FPGA. ping-pong RAM is to bridge the speed gap. There are logic control in it, determining the priority and sequence for reading and writing.

Block “transmitter” is in charge of starting from interleaving, then base-band filtering, modulating, all the data processing process before transmitting.

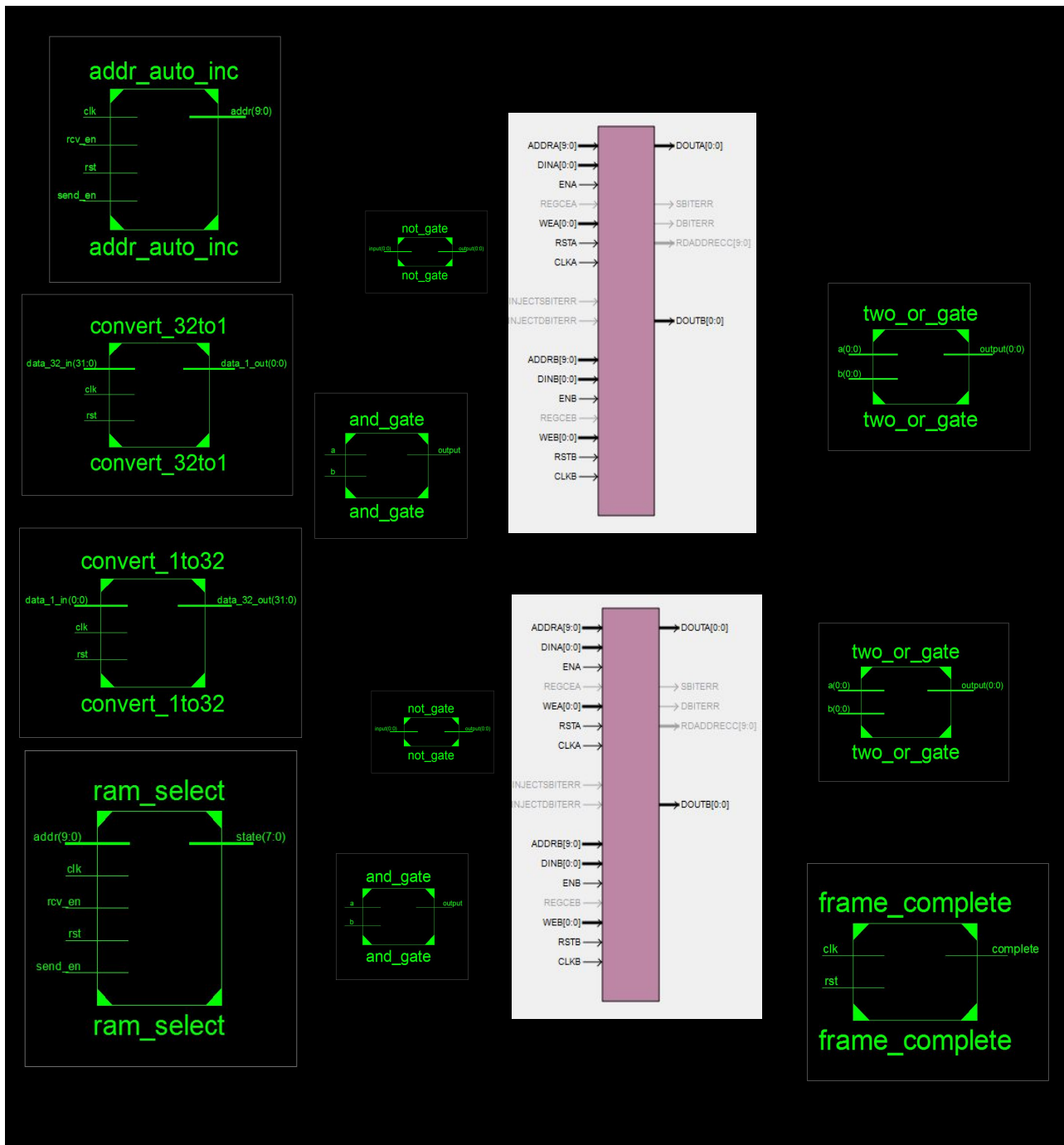
Block “TOP\_ADC” is to configure and control ADC ( Analog-Digital Converter) chip. By

writing serial control data into register, we can alternate the variables in ADC chip, adjusting its functions.

Block “TOP\_PLL” is to configure and control PLL ( Phase Locking Loop) chip. By writing serial control data into register, we can alternate the variables in PLL chip, adjusting its functions.

Block “receiver” undertakes the work starting from demodulation, all the way to reverse-interleaving, including filtering, acquisition, carrier restoration and tracking, etc.

See interconnect of block “emif\_top” below.



### Emif

A sequence in computer program is represented as a char array. Every single 0 or 1 is an element, in the form of a char variable. Each char occupies 8 bits in memory. Namely, 0 is 00000000 in memory; 1 is 00000001 in memory.

Emif interface can be configured to transmit 8 bits, 16 bits, 32 bits or 64 bits in parallel. In this system, we transmit 32 bit each time. In order to save storage space, data can be stored more densely. Four successive data could be stored in 31~24 bits, 23~16 bits, 15~8 bits and 7~0 bits respectively by bit shifting operation. In this project we do not adopt this practice, as the data

amount is not so large, the storage space is not tight. We just send 0 as 0X00000000 and send 1 as 0X00000001.

### Emif

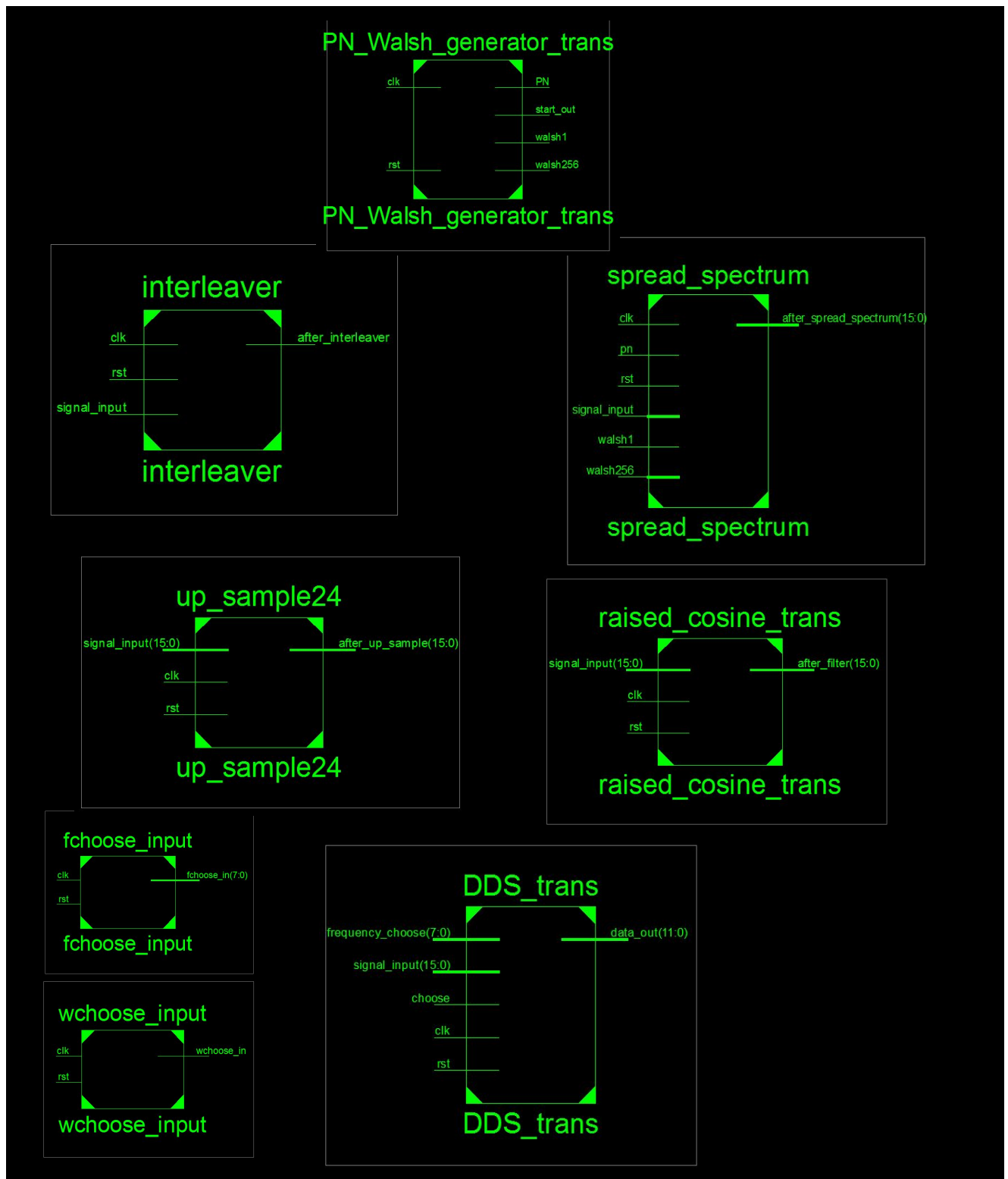
Emif is an interface to transmit data from DSP to FPGA. Specifically, data exchange is conducted through API ( Application Program Interface) function “EmifReadWrite” in CCS.

DSP arranged all the data it would like to send into an array. FPGA has four blocks of memory ( CE2~CE5) for receiving data. For example, we decide to use CS3. What we are supposed to do is to set a pointer “ pSyncData”, pointing to the base address of CS3.

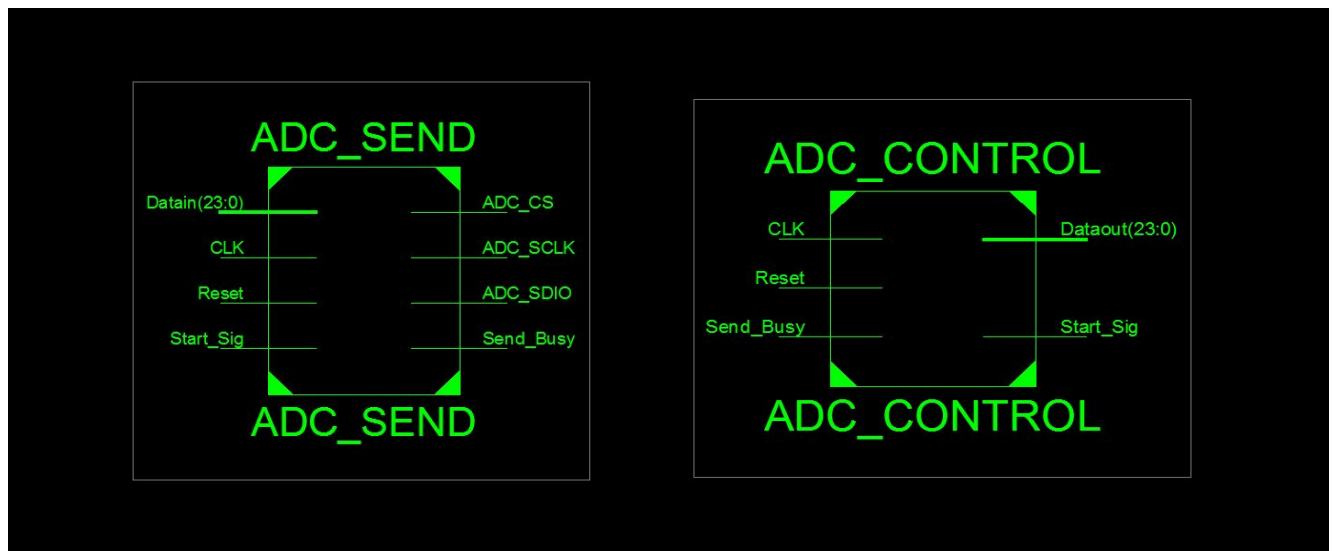
```
Uint32 *pSyncData = (Uint32 *)EMIFA_CE3_BASE_ADDR;
```

If DSP is to read data from FPGA, then this function starts from the base address ( in FPGA), packing the data into 32-bit-packages to send to DSP; if it is the DSP who want to write data into FPGA, then this function packs the data into 32-bit-packages, then writes them one by one into the FPGA, with the starting address of base address.

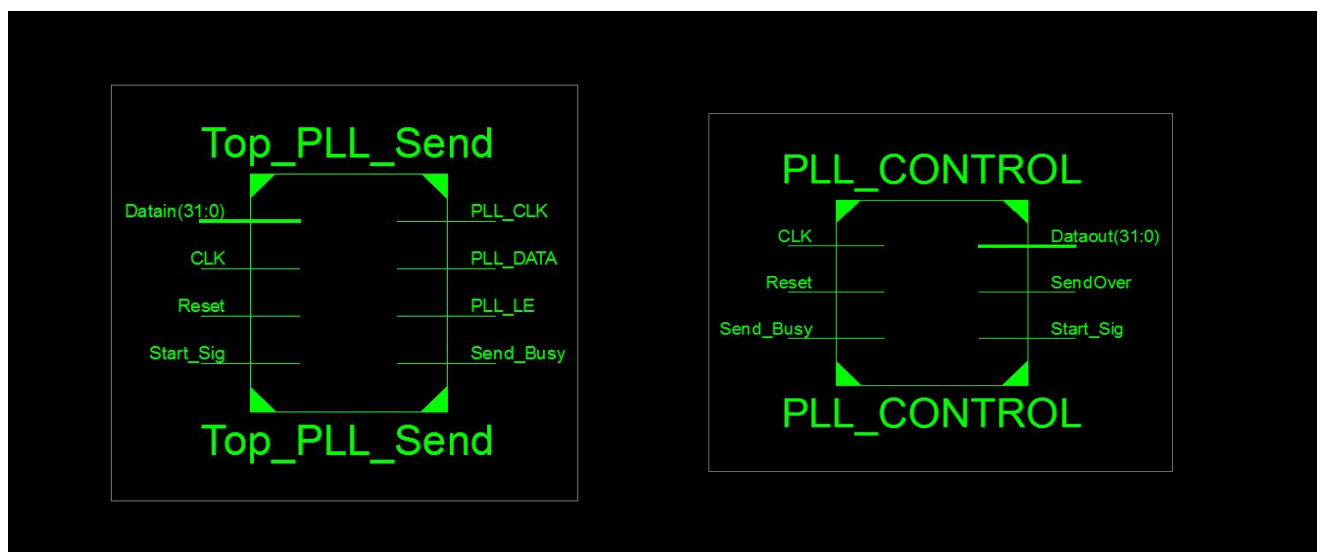
See interconnect of Block Transmitter below.



See interconnect of Block TOP\_ADC below.



See interconnect of Block TOP\_PLL below.



### Viterbi decoding

There are three common methods for decoding, algebra decoding, Viterbi decoding and sequential decoding. This system employs Viterbi decoding.

Recall the coding process first. We use convolution coding, at a constraint length of 9. That is, the current output is not only related to the current input, but also related to the 8 previous inputs. This fixed and rigorous relation among outputs can be represented in a state transition diagram (state machine is another alternative).

In this way, there exists a fixed relationship among the outputs of convolution coder. Errors could be detected and even revised thanks to this relationship. Viterbi decoding has a completed link of transfer. If a segment is off the link, we can tell there is something wrong.

The longer the constraint length, the higher the reliability for error detection. However, too long constraint length means a much large state transition diagram ( $2^N$  rows for the case of constraint length  $N$ ). There is no use to be so long since the reliability is high enough when constraint length is 9.

Viterbi decoding is conducted in DSP. We write C code in CCS ( Code Compose Studio) environment. Code Compose Studio environment focuses on the implement of algorithm abstractly, regardless of the binary mechanism in physical memory. Therefore we just regard it as array. Apart from the array method, it can also be represented in hexadecimal number, some of the operation can be done through bit shifting.

When it comes to the decision of symbols in receiver, we hope our decisive sequence to be the closest to the original sequence sent in transmitter. How to define the “close” ? How to assess the similarity of the two sequences ?

We introduce an indicator “distance” to measure the extent of similarity. Two typical measuring methods are Hamming distance and Euclidean distance, hard- decision and soft-decision correspondingly.

Hamming distance is the number of the inconsistent bits between two binary sequences. In our C code, it is calculated by function “TranDistance”.

```
unsigned int TranDistance(char *rcv, int s1_value, int s2_value)
```

This function calculates the distance between the receiving symbol “rcv” (2 bits long) and “the output symbol during the transfer from state  $s1$  to state  $s2$ ”. Specifically, set a variable “count”, then compare each bit, update the “count” by pulsing one when an inconsistent bit is found. If it is not reachable from state  $s1$  to state  $s2$ , output is 1000 as default. ( 1000 is a number large enough which we will not reach)

Euclidean distance is to view a sequence of length  $N$  as an  $N$ -dimension-vector. The value of Euclidean distance equals the distance of two vectors, applying the definition of vector distance.

$$\rho(A, B) = \sqrt{\sum_{i=1}^n (a[i] - b[i])^2}$$

Taking advantage of the “encapsulation” property of function, we can easily conduct soft-decision by just modifying internal code of it.

Function ”Viterbi” is to implement the core Viterbi algorithm.

```
void Viterbi( char *viterbiinput, char *viterbioutput)
```



Define each node in mesh (state transition diagram) as a construct.

```
struct stat //定义网格图中每一点为一个结构体
{
    unsigned int index; //即行号
    unsigned int measure; //转移到此状态累计的度量值
    unsigned int input_bit; //从前一状态转移到此状态的输入符号
    struct stat *last; //指向前一个状态的指针
};
```

state transition diagram (mesh):  $2^k \cdot (K-1) = 2^1 \cdot (9-1) = 256$ , 256 states in total ( constraint length 9).

```
struct stat state[256][MERGEDIST];
struct stat *state_ptr;
```

Transform the longer series “viterbi input” (384bits in total) into 192 two-dimensional-arrays.

```
char rcv[193][3];
for(i=0; i<193; i++)
{
    rcv[i][0] = viterbiinput[2*i];
    rcv[i][1] = viterbiinput[2*i+1];
    rcv[i][2] = '\0';
}
```

Initialize the measurement of each state :  $\text{state}[i][j].\text{measure} = 0;$

Initialize the index (index of row) of each state:  $\text{state}[i][j].\text{index} = i;$

set the initial state of “time 0” (i.e. the first column of the mesh):

```
state[m][0].measure = 1000;
state[m][0].input_bit = '0';
state[m][0].last = NULL;
```

Elaborate the state transition:

For the first 128 states, state p is transferred from state  $2 \cdot p$  and  $2 \cdot p + 1$ . The last input bit is 0.

```
unsigned int meas1 = state[2*p][t-1].measure + TranDistance( rcv[t-1], 2*p, p );
unsigned int meas2 = state[2*p+1][t-1].measure + TranDistance( rcv[t-1], 2*p+1, p );
```

The current accumulative measurement, equals to the sum of “the last accumulative measurement” and “ Hamming distance of output symbol from the last moment to the current symbol”.

Compare the two branches, take the one with smaller accumulative measurement.

```

//取其中累计衡量小的那条支路的各个参数 来作为该节点的参数
if( meas1 > meas2 )
{
    state[p][t].measure = meas2;
    state[p][t].input_bit = '0';
    state[p][t].last = &state[2*p+1][t-1];
}
else
{
    state[p][t].measure = meas1;
    state[p][t].input_bit = '0';
    state[p][t].last = &state[2*p][t-1];
}
}

```

For state ranging from 128 to 255 (128 states in total), they have the similar principle as the first 128 states. The state  $p$  comes from the state  $2*(p-128)$  and state  $2*(p-128)+1$ . The last input bit is 1.

```

//后 128 个状态都是由 2*(p-128) 和 2*(p-128)+1 状态转来的 且上一个输入比特为 1
for( p=128; p<256; p++ )
{
    unsigned int meas1 = state[2*(p-128)][t-1].measure + TranDistance( rcv[t-1], 2*(p-128), p );
    unsigned int meas2 = state[2*(p-128)+1][t-1].measure + TranDistance( rcv[t-1], 2*(p-128)+1, p );
}

```

Figure out the minimum measurement of at the moment after  $n$ th step, preparing for back routing. That is the reason why we store the previous state in each step. We can resume the route node by node.

```

//找出 n 步后度量值最小的状态, 准备回溯路由
measure_mim = state[0][MERGEDIST-1].measure;
printf("measure_mim = %d \n", state[0][MERGEDIST-1].measure );
back_ptr = &state[0][MERGEDIST-1]; // 每一步都找它上一步的指针

fclose(fp2);
fp3 = fopen ( "eee.txt" , "w" );
viterbioutput[MERGEDIST] = '\0';
for( u = MERGEDIST-1; u >= 0; u-- ) // 向前递归的找出最大似然路径
{
    viterbioutput[u-1] = back_ptr -> input_bit;
    //back_ptr 作为回溯的指针, 指向哪个状态图( 网格图) 节点就把那个节点的 value 值
    //( 输入符号) 存到 ViterbiOutput 数组中
    fprintf(fp3, "index:%d,measure[%d]:%d\n", back_ptr -> index, u, back_ptr -> measure);
    back_ptr = back_ptr->last; // 将指针向前移动一位
}
fputs("\n", fp3);
return;

```

After Viterbi decoding, the original frame before convolution coding is restored. To ensure the correctness, the decoded code should conduct CRC checking.

The 8 bit-CRC-checking bits are generated on the transmitter. The 144 bit message is divided by a generator polynomial. Then attach the remainder to the frame. For the receiver, it is supposed to

attach the received 8-bit-CRC-checking bits to the received message, then let it divided by the same generator polynomial. In ideal case, remainder should be 0. (however it will not happen most of the case)

The core task is to write a bit-shifting divider in C code.

```
for( j = 0; j < 144; j++ )
{
    for( i = 0; i <= 7; i++ ) //前8位(含最头补的那一位) 往左移位, 最高位自动丢弃
    {
        remainder[i] = remainder[i+1];
    }
    remainder[8] = message[j++]; //最后一位取message的新一位
    if ( strcmp( remainder, "101010111") < 0 ) //如果被除数的值比生成多项式的值小
    {
        continue; //则不进行操作 继续移位
    }
    else if ( strcmp( remainder, "101010111") >= 0 ) //如果被除数的值比生成多项式的值大
    {
        remainder[0] = ( remainder[0] + '1' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[1] = ( remainder[1] + '0' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[2] = ( remainder[2] + '1' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[3] = ( remainder[3] + '0' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[4] = ( remainder[4] + '1' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[5] = ( remainder[5] + '0' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[6] = ( remainder[6] + '1' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[7] = ( remainder[7] + '1' - 48 - 48 ) % 2 == 0? '0':'1';
        remainder[8] = ( remainder[8] + '1' - 48 - 48 ) % 2 == 0? '0':'1';
    }
}
```