ETIN01 IC-project & Verification, digital

OFDM Symbol Detection Report

Ma Ling (770821-8867)
Peyman Pouyan (831230-6296)

April 2009

INDEX

Abstract

OFDM is a popular technique nowadays. The symbol detection is a key issue of this technique. In this project, we implement a low complexity algorithm in hardware to detect the OFDM symbol, then report the result. The structure of this report is listed as follows.

In the first chapter, we introduce a low complexity algorithm of OFDM symbol detection and the two methods, Full Search and Random Search, to implement this algorithm in hardware.

In the following two chapters, we present project specification and the reference model which is carried out in Matlab （version R2007a 7.4.0.0287 ）.

In the fourth chpater, we describe the whole architecture, the blocks' function and the details about the implementaion of Full Search and Random Search. Then two methods' performance is compared through the verification in ModelSim.

In chapter five and six, we report the result of the synthesis and place and route

In the last two chapters, we make a conclusion and propose some further work.

1 Algorithm Introduction

1.1 OFDM Basics

OFDM which is the abbreviation of Orthogonal Frequency Division Multiplexing, is a standard multicarrier modulation for high data rate wireless and wired communication.

It has two important advantages which are spectral efficiency and multipath immunity and has been widely used for digital audio broadcasting(DAB),wireless local area network (WLAN) ,Digital Subscriber Loops (ADSL,VDSL) and other communication applications.

Fig1-1 shows a basic OFDM system and the part of our project in this system.

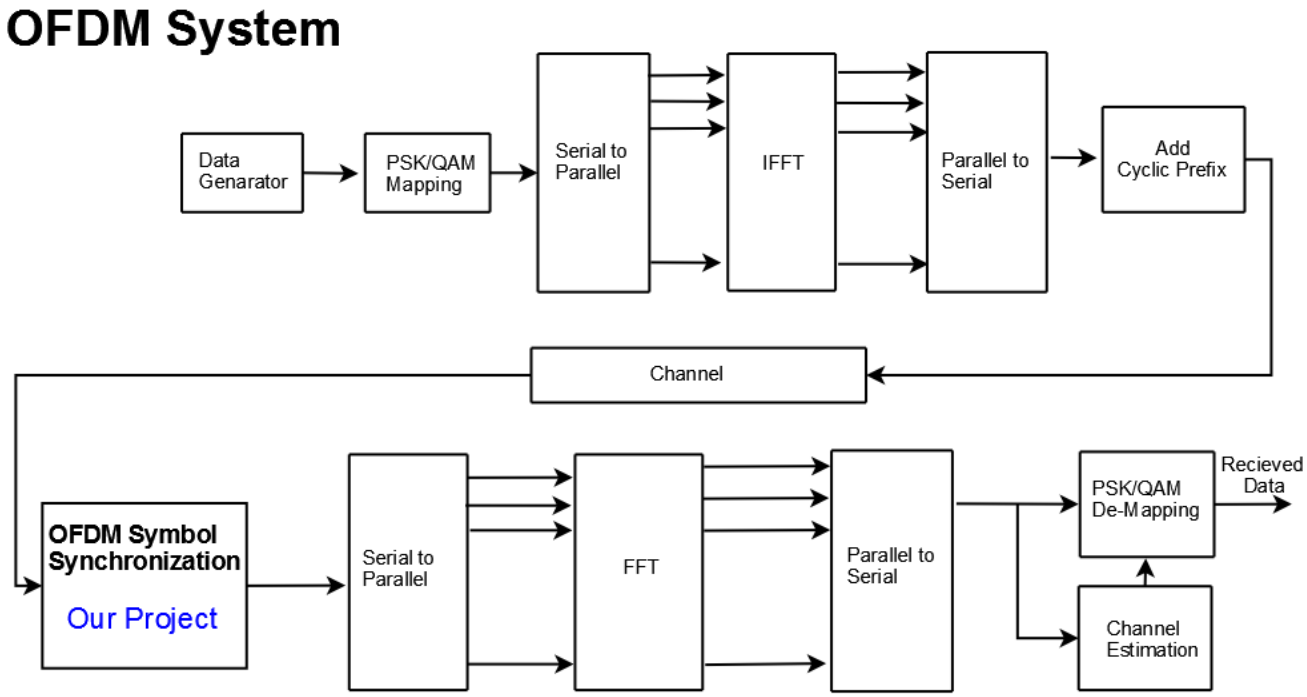


Fig1-1 OFDM system diagram

1.2 Synchronization Problem

OFDM has some drawbacks. One of them is the high sensitivity to time and frequency synchronization errors.

There are two sets of synchronization problems in OFDM, frequency error and timing error.

The frequency error is caused by mismatch between the transmitter and receiver oscillators and also the channel Doppler shift. And the timing error is caused by symbol timing and sample timing which is mostly caused by clock drift.

Fig1-2 shows the diagram of synchronization errors, the things we have detected in our project are symbol timing and frequency offset which are colored in green.

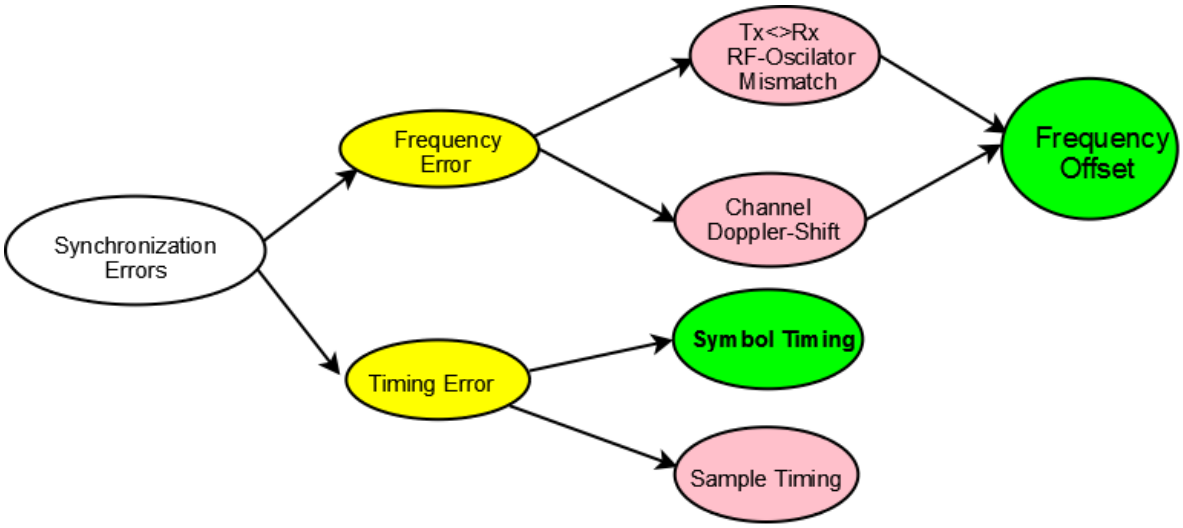


Fig1-2 synchronization errors

1.3 Synchronization Techniques

There are three major synchronization techniques described as below[4].

Pilot Symbols: here we use preamble in OFDM symbols. Its drawback is that lowers the achievable data rate.

Blind Synchronization Techniques: here we use cyclo-stationarity in OFDM signal. Its drawback is that is costly in computation.

Cyclic-Prefix Based Techniques: this method is efficient as we don't use pilot symbols and any way we need cyclic prefix for removing the multipath effect so they are freely available to be used for synchronization.In the next part we explain the cyclic-prefix method.

1.4 Cyclic-Prefix Based Techniques

Cyclic Prefix is a copy of last part of OFDM symbol to the beginning of the symbol and is removed at the receiver before demodulation. It should be at least as long as significant part of impulse response experienced by the transmitted signal.

In the following Fig1-3, Ncp is the number of cyclic prefix and N is number of samples. So one of our task in this project is to find the symbol start in OFDM data stream.This task is shown in the Fig1-4.

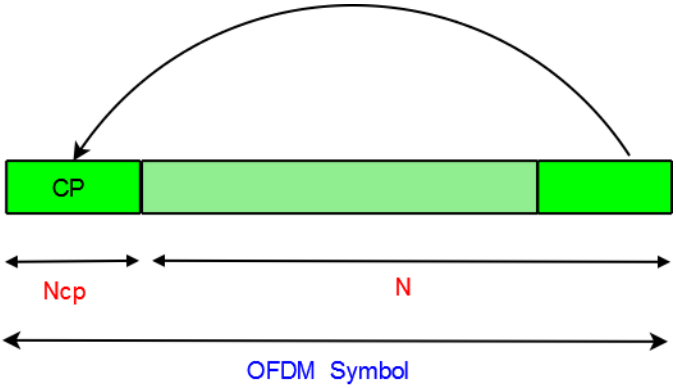


Fig1-3 OFDM symbol structure

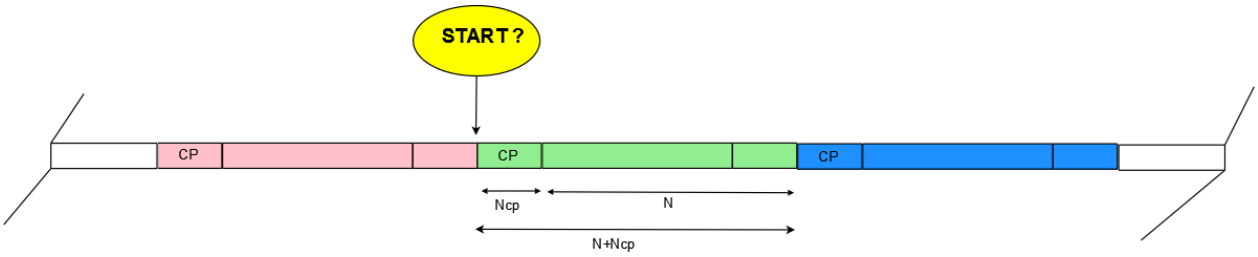


Fig1-4 symbol start in OFDM data stream

If the symbol start becomes wrong, it will lead to timing errors. Then we will lose the orthogonality of the OFDM signal and have Inter Symbol Interference(ISI) and phase error. We need to use some synchronization techniques to find the correct symbol start and carrier frequency offset.

There are two OFDM CP-based synchronization techniques which are implemented in our project.

1.4.1 CP-based synchronization algorithm (Algorithm1)

Algorithm1 is traditional CP-based synchronization algorithm. It depends on the calculation of the correlation function G(n) given by [1]

$$G(n) = \frac{1}{N_{cp}} \sum_{k=0}^{N_{cp}-1} r(n-k) r^*(n-k-N)$$

where r(n) represents the complex received signal samples, N is the number of subcarriers per OFDM symbol, and Ncp is the length of the cyclic prefix, and the asterisk * indicates the conjugate of a complex value.

This correlation function G(n) is used for both frequency and time synchronization. It is the correlation result of two sequences of Ncp samples length, separated by N samples, in the received sample sequence as shown in Figure 1-5.

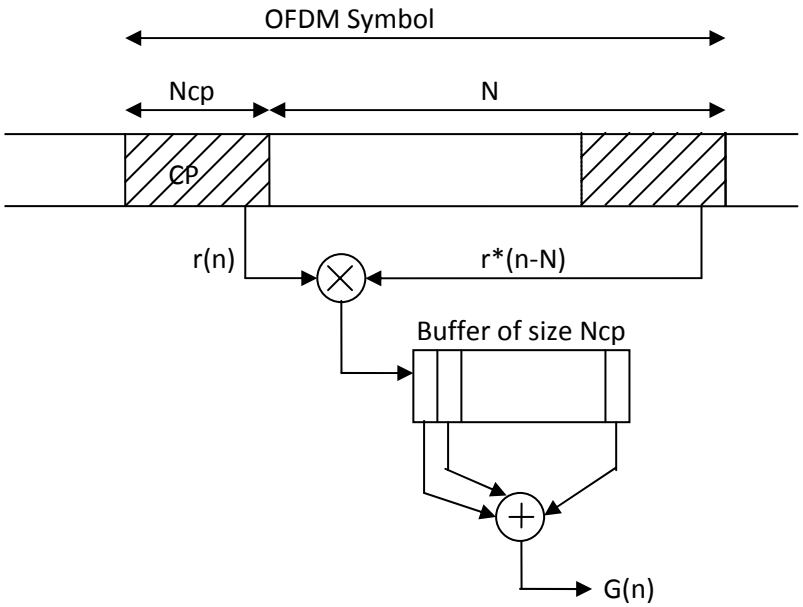


Fig1-5 the correlaton function G(n)'s computation

the estimation $\hat{T}_m$ of the symbol start for the mth OFDM symbol can be given as [1]

$$\hat{T}_m = \arg\max_{\theta\in\Theta} |G_m(\theta)|$$

The maximum of the correlation function is found over a window of $\Theta=\{\theta|1\le\theta\le Ncp + N\}$ for each OFDM symbol (window boundaries are not normally aligned with that of OFDM symbols).

And the frequency offset δf is estimated using the phase of the correlation function at $\theta = \hat{T}_m$ as given in [1],

$$\delta \hat{f}_m = -\frac{1}{2\pi} \angle G_m(\hat{T}_m)$$

This algorithm scans all points within the window. In our project, it is used for the first received symbol to find the best possible symbol timing which is the initiative point of algorithm2. In next part, we introduce algorithm2.

1.4.2 Low complexity algorithm (Algorithm2)

Algorithm2 is a low complexity algorithm.

If we use the number of complex multiplications to be as the measure of complexity, algorithm1 needs (N+Ncp)*Ncp complex multiplications per symbol. Based on discrete-stochastic approximation algorithms, a novel algorithm is given in [2], which can decrease the complexity.

Algorithm2 Steps:
i) Select a start point $\theta_0 \in \Theta$. Let $W_0(\theta_0)=1$ and $W_0(\theta)=0$ for all $\theta \in \Theta$, $\theta \neq \theta_0$, where the weight array $W_m(\theta)$ stores the number of times the point θ has been visited. Let m=0 and $\theta_m^*=\theta_0$. Move to next step.
ii) Generate a uniform random variable θ_m', where $\theta_m' \in \Theta$, $\theta_m' \neq \theta_0$. Move to next step.
iii) If $|G(\theta_m')|> |G(\theta_m)|$, let $\theta_{m+1}=\theta_m'$. Otherwise let $\theta_{m+1}=\theta_m$. Move to next step.
iv) Let m=m+1, $W_m(\theta_m)= W_{m-1}(\theta_m)+1$, and $W_m(\theta)= W_{m-1}(\theta)$ for all $\theta \in \Theta$, $\theta \neq \theta_m$. If $W_m(\theta_m)> W_m(\theta_{m-1}^*)$, let $\theta_m^*=\theta_m$. Otherwise, let $\theta_m^*=\theta_{m-1}^*$. Here, θ_m^* is the estimated symbol start of the current symbol. Move back to step ii).

Algorithm2 gives θ_m^* as the estimation of T. And the carrier frequency offset estimation is

$$\delta \hat{f} = -\frac{1}{2\pi} \angle G_m(\theta_m^*)$$

This algorithm only compares two points' correlation result within the window. For each symbol, there are 2*Ncp complex multiplications needed. So it's called low complexity algorithm. In our project, it is used for the received symbols to adjust the possible symbol timing and let the timing estimation converge to the accurate one.

1.4.3 Algorithm implemented in hardware

To implement the above algorithm in hardware, there are two key points.

Firstly, how to select the start point? There are two methods. One is using algorithm1 to pick out the most possible symbol start point. The combination of algorithm1 and algorithm2 is named as ***Full Search***. And another is just picking an arbitrary point within the window. We call it ***Random Search***. We implement, simulate both of them and compare their performance, which is stated in the chapter 4.

Secondly, how to generate the random number? Two ways are applied. One is using a ROM which contains the random numbers. These random numbers are generated a priori and written into ROM.
Another way is utilizing a LFSR component to generate the random number in real time. We realize two ways and compare their performance, which can be found in the chapter 4.

2 Project Specification

This OFDM symbol detection project has the following specification.

Coding Rate: 7/8

Phase Modulation: QPSK

FFT window(N)=512

Cp(Ncp)=32

Data Rate=2 Mbps

1 Frame=17 symbols

1 symbol=544 samples

3 Reference Model

3.1 Main functions

The reference model has three main functions.

a) Create real and imaginary data files of OFDM received symbols which have a zero symbol start and a certain frequency offset

b) Truncate some data in the first symbol to set a certain symbol start offset in the received symbols and keep the previous frequency offset

c) Implement Random Search (only algorithm2) and Full Search (algorithm1 and algorithm2). Fig3-1 shows the result of Random Search and Full Search in Matlab R2007a.

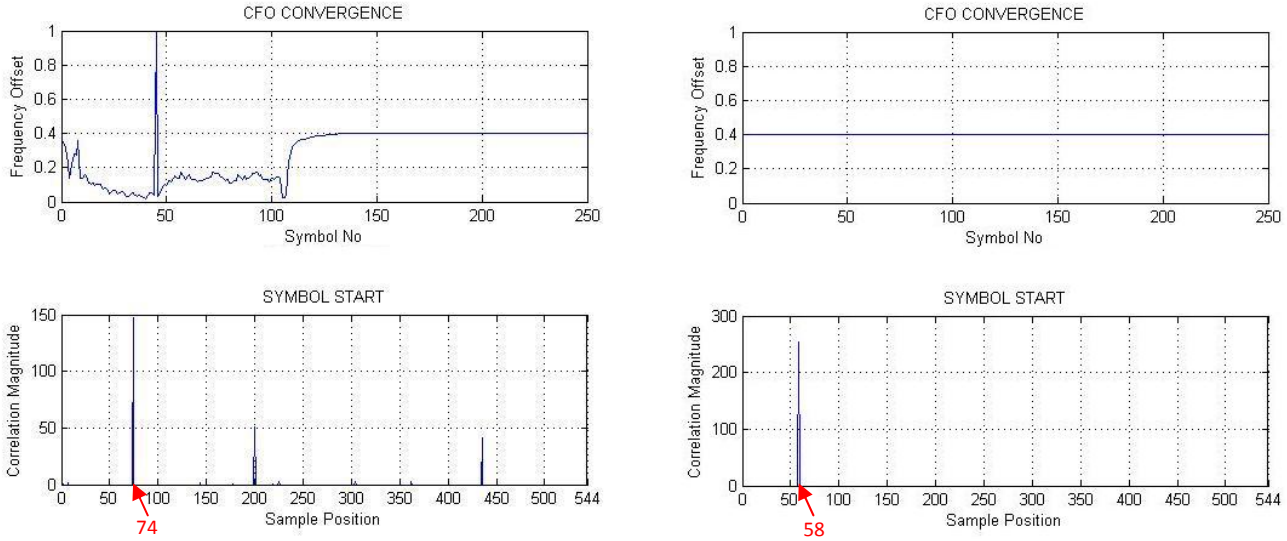


Fig 3-1 Random Search and Full Search Result in Matlab

The standard symbol start should be 58 and the CFO should be 0.4. From the above figures, we see that using Full Search can get a more accurate estimation than using Random Search.

3.2 Quantization error of input data

Since we create the 8 bits input data by scaling up and rounding the full precison data, the quantization error exists in this process.

Data in Binary (Data_b)	Normalized Data (Data_sim=$(Data\_b)_d$/127)	Full precison Data (Data_std)	Quantization error (Data_sim-Data_std)
10100110	-0.70866	-0.70711	-0.00155
00000000	0	0.003471	-0.003471
11111111	-0.007874	-0.0069419	-0.0009321
01011010	0.70866	0.70697	0.00169
00010000	0.12598	0.12772	-0.00174
11101011	-0.16535	-0.16507	-0.00028
11010111	-0.32283	-0.32411	0.00128
01001011	0.59055	0.59366	-0.00311
11001001	-0.43307	-0.43503	0.00196
11011100	-0.28346	-0.28019	-0.00327

Table 3-1 Input data quantization error

We calculate the quantization error which shows in Table 3-1 and get the mean value of the quantization error is -9.4231e-004, and the variance is 4.2496e-006.

4 RTL development

4.1 Architecture

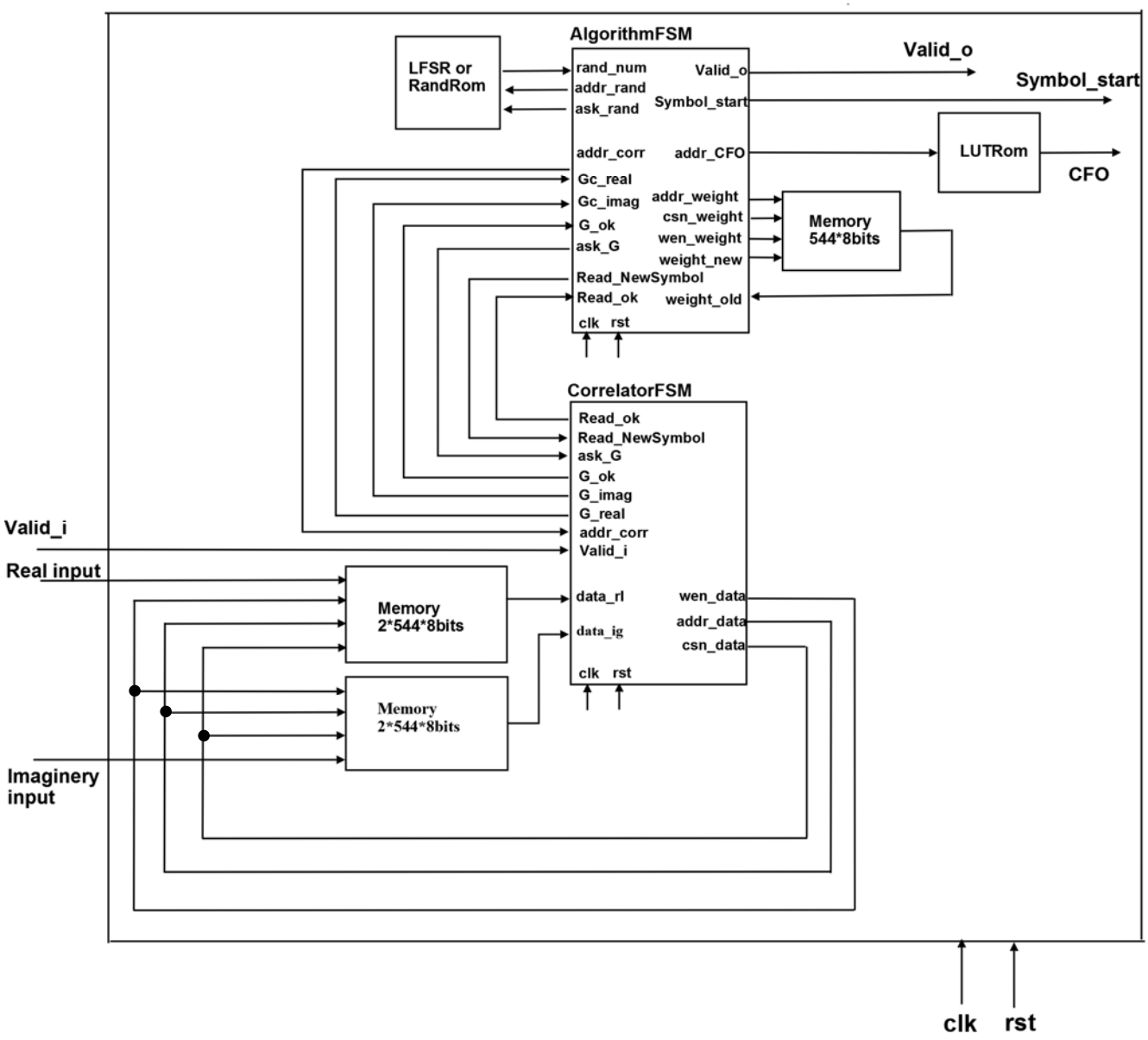


Fig 4-1 Architecture diagram

Input signals are real, imaginary data, clock, reset and valid_i.

Output signals are symbol start, carrier frequency offset and valid_o.

Our design contains four function blocks and three SRAMs.

a) AlgorithmFSM block implements Random Search (only algorithm2) and Full Search (algorithm1 and algorithm2) which are introduced in the chapter 1.4.3.

b) CorrelatorFSM block reads new symbol into SRAMs and executes complex correlation to get G(n).

c) Random number block generates random number whose range is from 1 to 544. It is a LFSR component or a ROM.

d) Frequency offset estimation block is a look up table which contains some possible frequency offset.

e) Two 1088 bytes SRAMs store real and imaginary part of input data. And one 544 bytes SRAM save weight information of each point within the window.

4.2 Blocks

4.2.1 AlgorithmFSM block

A Finite State Machine is used to implement Random Search (only algorithm2) and Full Search (algorithm1 and algorithm2). Which search is executed depends on whether the FullSearch state is included. As drawn in Fig4-2, if all in the red dotted frame is removed, Random Search is executed. Otherwise, Full Search is applied.

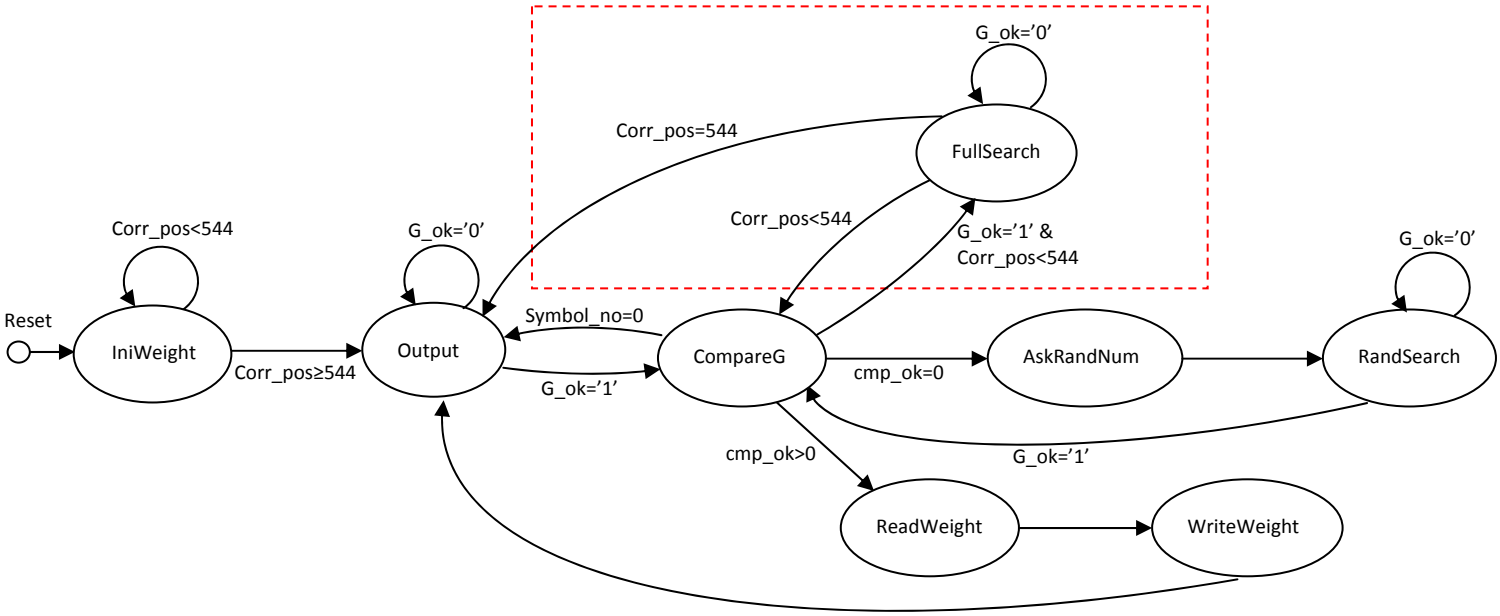


Fig 4-2 Algorithm's FSM

Each state in the above FSM is explained as follows.

a) IniWeight

When reset signal arrives, the system begins to run this state and initiates the SRAM which saves the weight data. We use 'Corr_pos' variable to counter how many units in the SRAM have been set to zero untill all the units are cleared. When 'Corr_pos' is larger than 543, it moves into Output state.

b) Output

In this state, the system outputs the estimated symbol start and frequency offset, sets ‘Corr_pos’ to zero, and waits for ‘G_ok’ signal which comes from CorrelatorFSM block. When ‘G_ok’ signal becomes high(G_ok=1), it jumps into CompareG state.

‘G_ok=1’ comes from CorrelatorFSM block and means the complex correlation has been calculated. If CorrelatorFSM block receives Read_NewSymbol signal, it also means that the new symbol has been stored into SRAMs.

c) CompareG

The correlation result is compared to find the bigger one in this state. The next state has four cases depending on different transition signals.

i) If Symbol_no=0, Output is the next state.

‘Symbol_no=0’ means there is only one symbol received and stored in the SRAM. The algorithms need at least two symbols to implement, so the system moves back to Output state and expects a new symbol arrived.

ii) If cmp_ok=0, the next state is AskRandNum.

‘cmp_ok=0’ shows the system only calculates the correlation result for last symbol’s estimated symbol position and a new position should be found out and sent to CorrelatorFSM block.

iii) When ‘cmp_ok’ is larger than zero, ReadWeight is the next state.

‘cmp_ok>0’ implies the comparison between two correlation results is finished.

iv) If G_ok=’1’ and Corr_pos<544, the system moves to FullSearch.

Here, ‘Corr_pos’ variable is used to mark the sample’s position which has been scanned.

d) AskRandNum

In this state, the system askes a new random number from the Random number generator, then jumps to RandSearch state.

e) RandSearch

A new estimated position is sent to CorrelatorFSM block and a high ‘G_ok’ signal will trigger the CompareG state.

f) ReadWeight

The position which corresponds with the larger correlation magnitude is selected as the symbol start position of the current symbol. Then the system reads the old weight at this position from the SRAM and moves to WriteWeight state.

g) WriteWeight

At this state, the system updates the weight data for the current symbol start position, generates the address of frequency offset look up table, sends Read_NewSymbol signal to CorrelatorFSM and jumps to Output state.

h) FullSearch

The correlation of each sample position in the first symbol is calculated to find the maximum value. The correspondent point is selected as the starting point. After all the sample positions are scanned (Corr_pos=544), Output is the next state. Otherwise the system moves to CompareG state.

4.2.2 CorrelatorFSM block

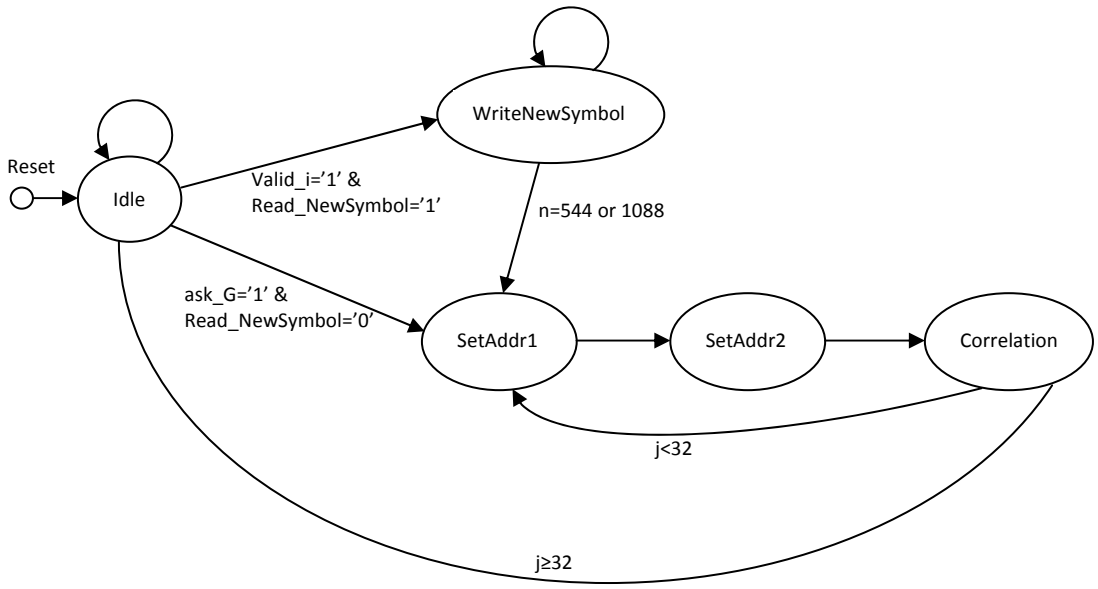


Fig 4-3 Correlator's FSM

Each state in the above FSM is explained below.

a) Idle

When reset signal arrives, the system stays at the Idle state and waits for the Valid_i signal which means the input data is ready. While Valid_i = '1', Read_NewSymbol signal from AlgorithmFSM block (Read_NewSymbol='1') drives the system into

WriteNewSymbol state. Otherwise when ask_G = ‘1’ and Read_NewSymbol=’0’, the system moves to SetAddr1 state.

Valid_i = ‘1’ shows that the input data is ready.

Read_NewSymbol=’1’ means new symbol is needed to implement the algorithm.

ask_G = ‘1’ implies that the complex correlation at a new position should be calculated.

b) WriteNewSymbol

At this state, new symbol is written into SRAMs. Then the system enteres SetAddr1 state and begins to calculated the correlation at the previous estimated symbol start point of the new symbol.

c) SetAddr1

The system sets the address for the data at the correlation’s first part, then moves to SetAddr2 state.

d) SetAddr2

The system sets the address for the data at the correlation’s second part and jumps to Correlation state.

e) Correlation

The system calculates the correlation according to the correlation position from AlgorithmFSM block. When the correlation isn’t finished (j<32), the next state is SetAddr1 state and set the address for the following data. Otherwise, Idle state should be the next state.

4.2.3 Look up table

We choose the highest 5 bits of real and imaginary data of the correlation result to form a 10 bits address. For each address, the carrier frequency offset(CFO) is calculated accordingly.

In the calculation, there are six cases considering the sign of the real and imaginary data, which is given in the following table.

Index	Gc_imag	Gc_real	CFO
1	>0	>0	2π-arctan(Gc_imag/Gc_real)
2	<0	>0	-arctan(Gc_imag/Gc_real)
3	≠0	<0	π-arctan(Gc_imag/Gc_real)
4	<0	=0	0.25
5	>0	=0	0.75
6	=0	any	0.5

Table 4-1 CFO's computation formula

The output is scaled by 100. For example, output 25 means the carrier frequency offset is 0.25.

Since only highest 5 bits are selected to compute the CFO, this look up table doesn't cover all the possible value and the quantization error exists in this process. Sometimes, the CFO is an approximation.

Real part (Gc_real)	Imaginary part (Gc_imag)	CFO in LUT (CFO_lut)	CFO standard (CFO_std)	Quantization error (CFO_lut-CFO_std)
-40251	-24907	0.4118	0.38	-0.0318
-59743	-34795	0.4161	0.38	-0.0361
-65018	-47595	0.3994	0.38	-0.0194
-87331	-86014	0.3762	0.38	0.0038
-95130	-82835	0.3860	0.38	-0.006
-130375	-95027	0.3998	0.38	-0.0198
-169669	-123357	0.3999	0.41	0.0101
-177519	-120079	0.4053	0.41	0.0047
-181345	-136204	0.3975	0.38	-0.0175
-201144	-154975	0.3955	0.4	0.0045

Table 4-2 CFO quantization error

We calculate the quantization error in Table 4-2 and get the mean value of the quantization error is -0.0108, and the variance is 2.6981e-004.

4.2.4 Random number generator

For implementing random search in our project, we need an uniform distribution of random numbers between 1 to 544. At first, we used a Rom with 256 random

numbers. After completing our project, we decide to generate our random numbers in real time. So we tried to find different methods of genarating random numbers in FPGAs. By reading Xilinx application notes[3], we find out that one of the regular ways for generating random numbers is using a LFSR which can generate pseudo-random numbers with uniform distribution. We introduce the above two ways in the following part.

a) use a rom to provide random number to AlgorithmFSM block.

We generate a rom whose address is 8 bits and increased from 0 to 255 by step 1. The output has 10 bits length and varies from 1 to 544.

Due to the memory size limitation, this ROM doesn't contain all the numbers from 1 to 544. So the Random Search can't get an accurate symbol start if Full Search isn't applied.

b) LFSR

LFSR has finite number of possible states, and after a fixed period, it enters a repeating cycle[5]. We can have a LFSR with a well-chosen feedback function that can produce a sequence of bits which appears random and covers enough numbers.

In LFSR, which stands for linear feedback shift register is a shift register whose input bit is a linear function of its previous state.The linear function of single bit is XOR or XNOR.

LFSR sequence through 2^N-1 states that N is number of registers.At each clock edge the contents of the registers are shifted right by one position and the empty bit is filled with a bit which is a feedback from XOR or XNOR of some special registers .

These special registers are called taps and choosing these taps carefully can get a long random number sequence which covers all numbers between 1 and 544.

Fig4-4 shows a 4-bits LFSR whose taps are 3 and 4. We use a10-bits LFSR whose taps are 7 and 10.

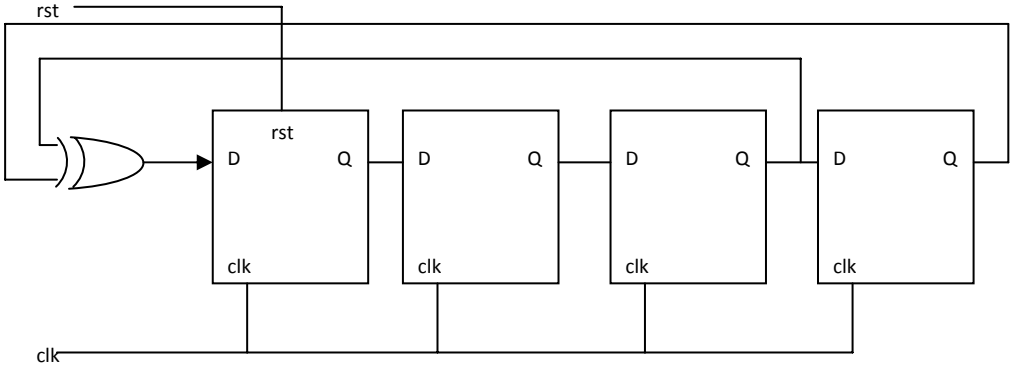


Fig4-4 4-bits LFSR Sketch

Theoretically, if there is enough long time for LFSR to generate all the numbers between 1 and 544, the Random Search can get an accurate symbol start even if Full Search isn't applied.

4.3 Verification

Because we implement Random Search (only algorithm2) and Full Search (algorithm1 and algorithm2) in AlgorithmFSM block and realize random number ROM and LFSR for random number generator, there are four designs composed by these components. We verify all of them.

4.3.1 ModelSim result

10 sets of input data are created by reference model, which contain 256 symbols and have standard timing (symbol start) and CFO.

We use two letters to indicate the design. The first letter represents Full Search (FS) and Random Search (RS). The second letter denotes LFSR (L) and random number ROM (R). For example, Timing FS-L means the estimated symbol start of the design which uses Full Search and LFSR.

Timing standard	CFO standard	Timing FS-L	CFO FS-L	Timing FS-R	CFO FS-R	Timing RS-L	CFO RS-L	Timing RS-R	CFO RS-R
5	0.9	5	0.91	5	0.91	0	0.88	5	0.88
7	0.2	7	0.21	7	0.21	0	0.21	5	0.21
58	0.4	58	0.4	58	0.4	60	0.38	61	0.41
170	0.25	170	0.25	170	0.25	162	0.25	171	0.25
256	0.8	256	0.8	256	0.8	261	0.82	250	0.8
311	0.5	311	0.5	311	0.5	313	0.5	315	0.46
382	0.75	382	0.75	382	0.75	65	0.7	380	0.7
467	0.85	467	0.84	467	0.84	462	0.8	463	0.82
521	0.65	521	0.63	521	0.63	522	0.59	523	0.59
543	0.1	543	0.13	543	0.13	0	0.13	0	0.16

Table 4-3 Simulation's result

The above table shows that using Fullsearch to estimate can get an accurate symbol start. Sometimes the frequency offset estimation is not very accurate, marked in ▨, due to the Look up table limitation which doesn't include all the possible datas. Using Randsearch to estimate can't get an accurate symbol start, marked in ▨, due to the random number generator. If the random number generator can't cover all the numbers between 1 and 544, we can't get an accurate result.

4.3.2 ModelSim waves

The following waves are the result when we use the input data whose standard symbol start is 58 and CFO is 0.4.

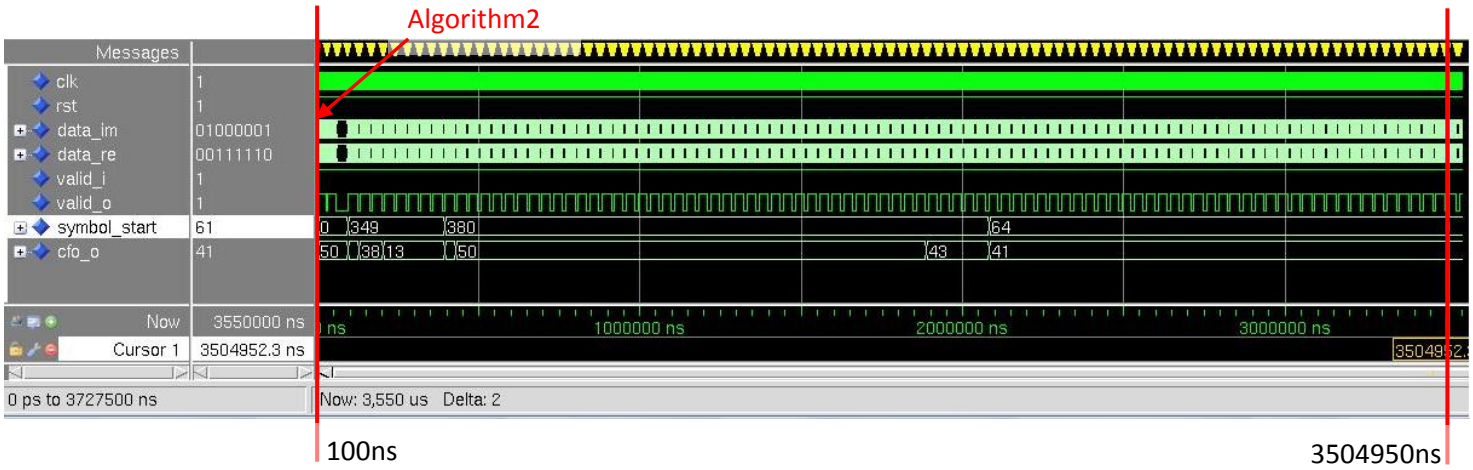


Fig4-5 Random Search simulation result

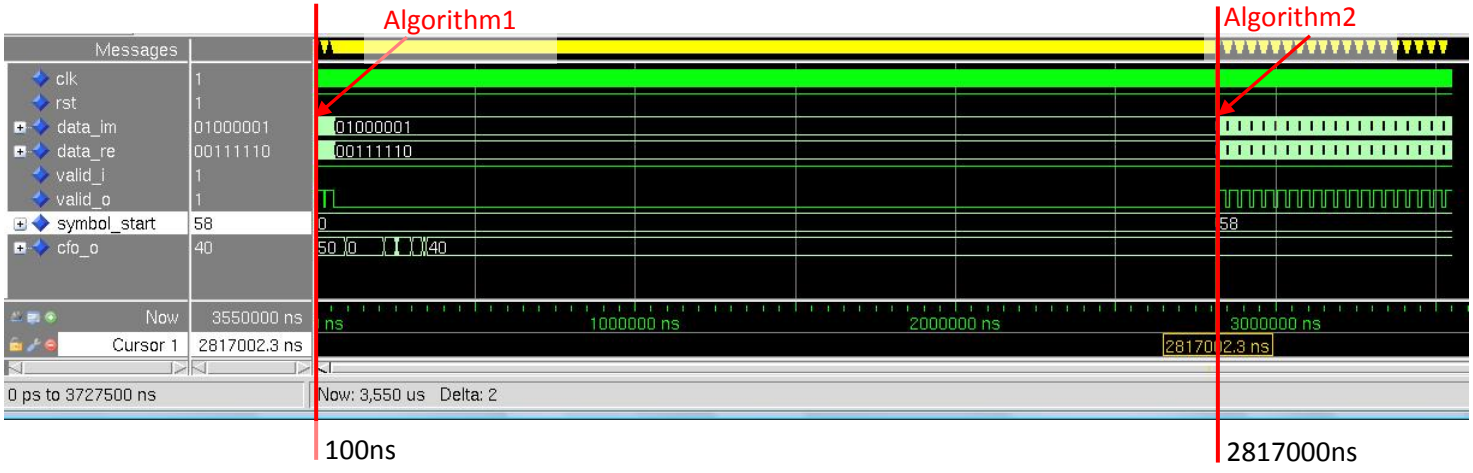


Fig4-6 Full Search simulation result

We set the clock frequency as 20MHz, reset system after 100ns and run the simulations 10 times and observe that though Full Search needs a fixed period, 2816900ns, to execute algorithm1 before implementing algorithm2, it spends less

time to get an accurate timing estimation than Random Search which only runs algorithm2 and takes 3504850ns.

But sometimes Random Search needs less time to converge to an inaccurate result, because convergence time is a random process for Random Search.

4.3.3 Achievable Input Data Rate

The current clock frequency is 20MHz. To calculate the input data rate under Full Search and Random Search, we observe when the new symbol arrives at the ModelSim. The following two figures depict the symbol arrived time when carring out Full Search and Random Search.

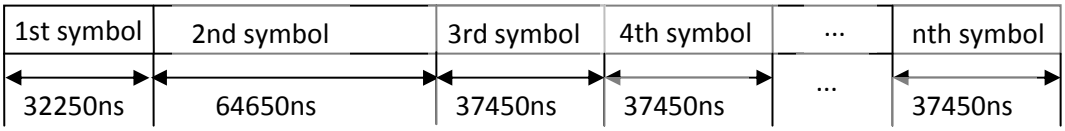


Fig4-7 Symbol arrived time under Random Search

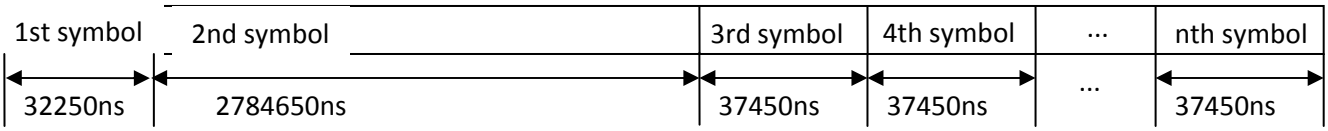


Fig4-8 Symbol arrived time under Full Search

So the input data period should be larger than 64650ns (1293 clock cycles) when using Random Search method. For Full Search, the data period must not less than 2784650ns (55693 clock cycles), which is slower than Random Search's data period. From the third symbol, both the methods enter the stable state. At this state, the input data period can be reduced to 37450ns (749 clock cycles).

5 Synthesis

Since Full Search can get an accurate result, we choose FS-L design which uses Full Search and LFSR, and FS-R design which uses Full Search and Random Rom, to do the Synthesis. FS-LA and FS-RA is for Min Area, and FS-LH and FS-LH is for High speed.

5.1 Constraints and Result

a) Synthesizing for Min Area:

create_clock "clock" -period 20 -name clock
set_clock_uncertainty 0.1 clock
set_max_area 0
compile -map_effort high
set_max_area **189172** (Design with LFSR)
compile -map_effort high

Result:

	FS-LA	FS-RA
Data required time	19.83	19.83
Data Arrival time	14.12	14.12
Slack	5.71	5.71

b) Synthesizing for High Speed:
We synthesis the design to find a smallest clock period which won't cause any violation. And the result shows the smallest clock period can reach 8ns.

create_clock "clock" -period 0 -name clock
set_clock_uncertainty 0.1 clock
compile -map_effort high

Result:

	FS-LH	FS-RH
Data required time	7.79	7.77
Data Arrival time	7.79	7.77
Slack	0	0

5.2 Area Comparison

	FS-LA	FS-RA	FS-LH	FS-RH
AlgorithmFSM	29665	29336	38644	34242
CorrelatorFSM	15048	15048	18208	16640
Random Generator	347(LFSR)	2943(Rom)	337(LFSR)	2549(Rom)
LUTRom	5166	5166	4900	4988
SRAMs	133230	133230	133230	133230
Pads	5716	5716	5716	5716
Combinational area	44119	46930	55981	52853
Noncombinational area	145053	144509	145054	144512
Total cell area	189172	191439	201035	197365

5.3 Design's critical path and Schematic

Critical Path is shown below for clock period of 20ns.

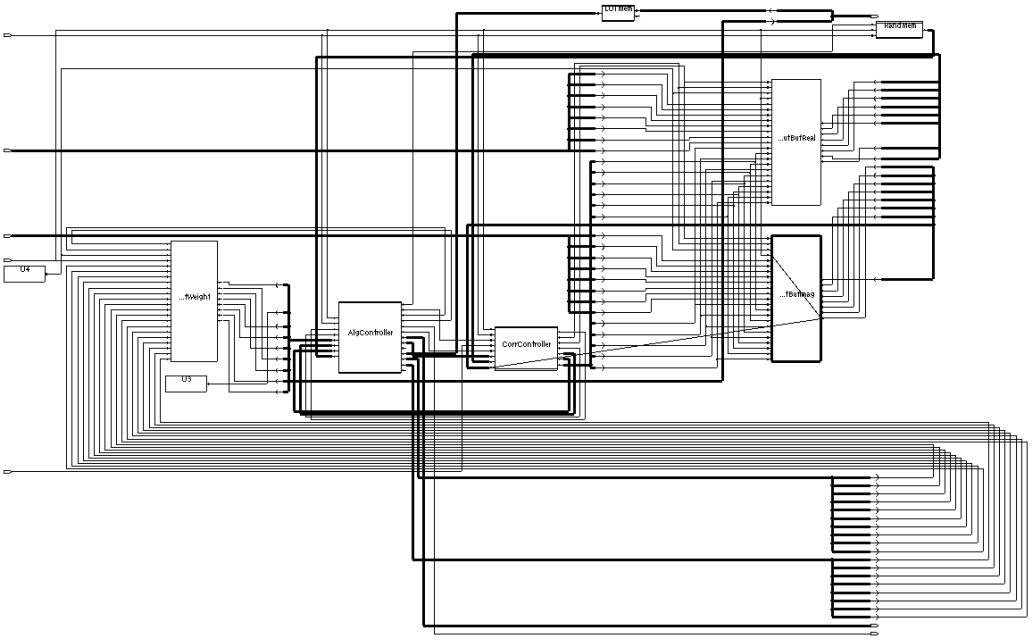


Fig5-1 FS-LA's Critical Path

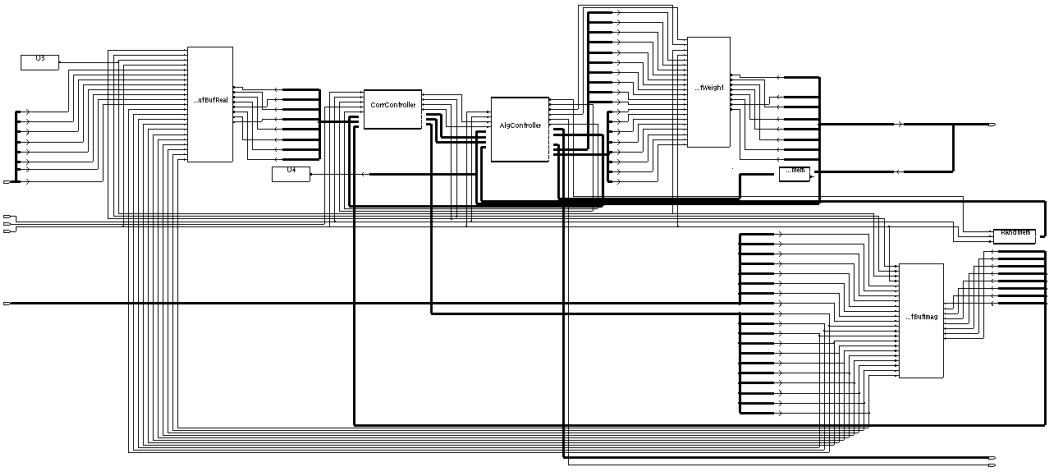


Fig5-2 FS-LH's Critical Path

FS-LA and FS-RA have the same critical path, as shown in Fig5-1. Imaginary SRAM, CorrelatorFSM and AlgortithmFSM blocks are in the critical path. The lowest bit of imaginary data causes the maximum delay.

FS-LH and FS-RH also have the same critical path, as drawn in Fig5-2.The blocks in the critical path are Real SRAM, CorrelatorFSM and AlgorithmFSM blocks. The lowest bit of real data leads to the delay.

5.4 Design's maximum clock frequency

Our maximal clock frequency is achieved for clock period of 8ns.

	FS-LH	FS-RH
Frequency	125 MHz	125 MHz

From the analysis of chater 4.3.3, we can estimate the achievable input data rate if the clock frequency is 125MHz.

For Random search, the input data period is at least 10344ns, so the data rate is around 1.54Mbps. For Full search, the input data period is not less than 445544ns, so the data rate is around 35.91Kbps. At the stable state, the input data period is reduced to 5992ns, then the achievable data rate can be increased to more than 2.67Mbps.

5.5 Two designs' comparison

We can see that FS-L (Full Search with LFSR) costs less area than FS-R (Full Search with Random ROM), but the two designs have the same critical path and can reach the same highest speed.

6 Place and Route

6.1 Core Area.

Synthesis:

Total cell area: 189172 um^2

Place & route:

Total area of Standard cells: 68240.640 um^2

Total area of Standard cells (Subtracting Filler Cells): 68240.640 um^2

Total area of Macros: 170533.760 um^2

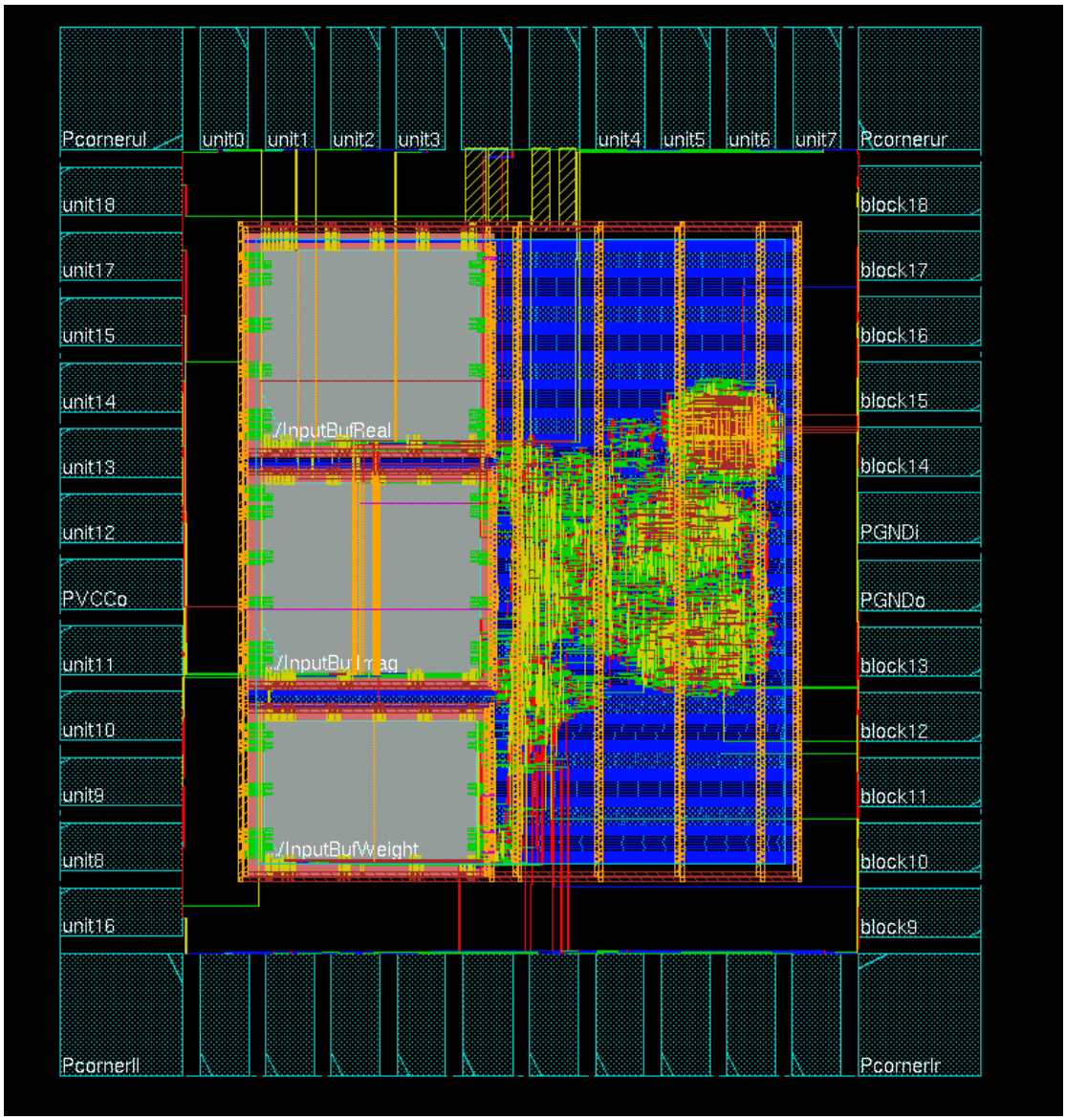
Total area of Blockages: 0.000 um^2

Total area of Pad cells: 495520.000 um^2

Total area of Core: 503801.824 um^2

Total area of Chip: 1472992.104 um^2

6.2 Layout Figure



6.3 Violations

We had Violation in our pads.The first time we had filled our pads and we had the violation which tells us we have overlap in our pads.The second time we did not fill them and we did not have any violations.

7 Conclusion

The previous chapters shows the issues of OFDM synchronization, the algorithm executed on FPGA, the project specification, reference model, the design architecture, the details of implementation, and the result of simulation, synthesis, and Place and Route.

We can get a conculsion that the low complexity algorithm is implemantable on FPGA, and using Full Search to detect the OFDM symbol gives more accurate result than using Random Search, but its achievable input data rate is slower than Random Search's.

8 Futher work

There are three aspects which can be improved in the future work.

a) For Random Search method, its disadvantage is the inaccurate estimation result.

Since this result is very close to the accurate one, we can do an additional search in a small range to find the accurate result. Which algorithm is suitable for the additional serach should be studied more.

b) For Full Search method, its drawback is the low input data rate.

Because the fixed period make the input data rate slow, we can use an extra SRAM to store the data at this period when increasing the data rate. But it takes large area. A tradeoff should be done between the input data rate and chip area.

c) For Look-up table, it outputs the carrier frequency offset with low accuracy.

Due to the truncation of correlation magnitude, the estimated frequency offset losts some precision. A better algorithm to calculate the accurate frequency offset can be explored.

Appendix

P&R Script-files

```
#############################################################
#  Encounter Command Logging File                  #
#  Created on Thu Apr 16 17:34:45                  #
#############################################################
loadConfig /h/d3/r/sx08lm4/OFDM/soc/Default.conf 0
commitConfig
fit
setDrawView fplan
setObjFPlanBox Instance OFDM_component/InputBufReal 161.3 628.656 428.5 863.056
uiSetTool move
deselectAll
setObjFPlanBox Instance OFDM_component/InputBufImag 163.141 362.335 430.341 596.735
uiSetTool move
deselectAll
setObjFPlanBox Instance OFDM_component/InputBufWeight 163.47 155.284 426.67 327.284
uiSetTool move
deselectAll
setObjFPlanBox Module OFDM_component 598.694 454.0 882.294 847.6
uiSetTool move
floorPlan -site core -r 1.18785040971 0.476806 90.0 110.0 90.0 110.0
setRoutingStyle -top -style m
uiSetTool select
fit
addHaloToBlock 20 20 20 20 -allBlock
cutRow
clearGlobalNets
globalNetConnect VCC -type pgpin -pin VCC -inst *
```

globalNetConnect VCC -type tiehi -pin VCC -inst *

globalNetConnect GND -type pgpin -pin GND -inst *

globalNetConnect GND -type tielo -pin GND -inst *

addRing -spacing_bottom 1 -width_left 4.9 -width_bottom 4.9 -width_top 4.9 -spacing_top 1 -layer_bottom metal5 -width_right 4.9 -around core -offset_bottom 10 -layer_top metal5 -option_file /h/d3/r/sx08lm4/OFDM/soc/appOption.dat/ofdm.ppo.ppo -offset_left 10 -spacing_right 1 -spacing_left 1 -offset_right 10 -offset_top 10 -layer_right metal6 -nets {GND VCC } -layer_left metal6

addRing -spacing_bottom 1 -width_left 4.9 -width_bottom 4.9 -width_top 4.9 -spacing_top 1 -layer_bottom metal5 -width_right 4.9 -around each_block -offset_bottom 10 -layer_top metal5 -option_file /h/d3/r/sx08lm4/OFDM/soc/appOption.dat/ofdm.ppo.ppo -offset_left 10 -spacing_right 1 -spacing_left 1 -type block_rings -offset_right 10 -offset_top 10 -layer_right metal6 -nets {GND VCC } -layer_left metal6

addStripe -set_to_set_distance 100 -spacing 1 -option_file /h/d3/r/sx08lm4/OFDM/soc/appOption.dat/ofdm.ppo.ppo -direction vertical -layer metal6 -stop_x 895.028 -width 4.9 -nets {GND VCC } -start_x 559.405

setCTSMode -traceDPinAsLeaf false -traceIoPinAsLeaf false -routeClkNet false -routeGuide true -topPreferredLayer 4 -bottomPreferredLayer 3 -routeNonDefaultRule {} -useLefACLimit false -routePreferredExtraSpace 1 -opt true -optAddBuffer false -moveGate true -useHVRC true -fixLeafInst true -fixNonLeafInst true -verbose false -reportHTML false -addClockRootProp false -nameSingleDelim false -honorFence false -useLibMaxFanout false -useLibMaxCap false

getClockMeshMode -quiet

setFillerMode -reset

setFillerMode -corePrefix FILLER -createRows 1 -doDRC 1 -deleteFixed 1 -ecoMode 0

setNanoRouteMode -quiet -routeBottomRoutingLayer 1

setNanoRouteMode -quiet -routeTopRoutingLayer 8

setNanoRouteMode -quiet -drouteEndIteration default

setNanoRouteMode -quiet -droutePostRouteWidenWireRule NA

setNanoRouteMode -quiet -drouteStartIteration default

setOptMode -effort high -leakagePowerEffort none -yieldEffort none -simplifyNetlist false -setupTargetSlack 0 -holdTargetSlack 0 -maxDensity 0.95 -drcMargin 0 -usefulSkew false

setPlaceMode -reset

setPlaceMode -congEffort medium -timingDriven 1 -modulePlan 1 -doCongOpt 0 -clkGateAware 0 -powerDriven 0 -ignoreScan 1 -reorderScan 1 -ignoreSpare 1 -placeIOPins 1 -moduleAwareSpare 0 -checkPinLayerForAccess { 1 } -preserveRouting 0 -rmAffectedRouting 0 -checkRoute 0 -swapEEQ 0

setScanReorderMode -reset

setScanReorderMode -skipMode skipNone -clkAware false -defInForce false -allowSwapping false -keepHierPorts false

setStreamOutMode -specifyViaName default -SEvianames false -virtualConnection true -uniquifyCellNamesPrefix false

setTieHiLoMode -reset

setTieHiLoMode -honorDontTouch false

setTrialRouteMode -highEffort false -floorPlanMode false -detour true -maxRouteLayer 8 -minRouteLayer 1 -handlePreroute false -autoSkipTracks false -handlePartition false -handlePartitionComplex false -useM1 false -keepExistingRoutes false -ignoreAbutted2TermNet false -pinGuide true -honorPin false -selNet {} -selNetOnly {} -selMarkedNet false -selMarkedNetOnly false -ignoreObstruct false -PKS false -updateRemainTrks false -ignoreDEFTrack false -printWiresOnRTBlk false -usePagedArray false -routeObs true -routeGuide {} -blockageCostMultiple 1 -maxDetourRatio 0

placeDesign -prePlaceOpt

setDrawView place

clockDesign -specFile design.cts -outDir clock_report -fixedInstBeforeCTS

getFillerMode -quiet

findCoreFillerCells

addFiller -cell FILLER8EHD FILLER64EHD FILLER4EHD FILLER3HD FILLER32EHD FILLER2HD FILLER1HD FILLER16EHD FILLER8ELD FILLER64ELD FILLER4ELD FILLER3LD FILLER32ELD FILLER2LD FILLER1LD FILLER16ELD -prefix FILLER –markFixed

sroute -noPadRings -jogControl { preferWithChanges differentLayer }

setNanoRouteMode -quiet -routeBottomRoutingLayer 1

setNanoRouteMode -quiet -routeTopRoutingLayer 8

setNanoRouteMode -quiet -drouteEndIteration default

setNanoRouteMode -quiet -routeWithSiDriven false

trialRoute -handlePreroute

setCteReport

writeDesignTiming .timing_file.tif

routeDesign -globalDetail

verifyGeometry

violationBrowser -all -no_display_false

Acknowledgments

Thanks to Joachim Rodrigues, Isael Diaz, Deepak Dasalukunte,Chenxin Zhang and Johan Löfgren for helping us to complete our project. It's our pleasure to study with your group.

Reference

[1] T. Keller, L. Piazzo, P. Mandarini, and L. Hanzo, "Orthogonal frequency division multiplex synchronization techniques for frequency selective fading channels", IEEE Journal on Selected Areas in Communications, Vol. 19, No. 6, June 2001, pp. 999-1007.
[2] Athaudage, C.R.N. and Krishnamurthy, V. "A low complexity timing and frequency synchronization algorithm for OFDM systems", Global Telecommunications Conference, Volume 1, 17-21 Nov. 2002, Page(s):244 – 248.
[3] Peter Alfke. "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators", XAPP 052 July 7, 1996 (Version 1.1), Xilinx.
[4] Chandra Athaudage. "OFDM for Wireless Multimedia Communications Synchronization & Other Issues", ARC Special Research Centre for Ultra-Broadband Information networks.
[5] http://en.wikipedia.org/wiki/Linear_feedback_shift_register
[6] IC-project & Verification , digital.
http://www.eit.lth.se/index.php?id=241&ciuid=197&L=1