國立交通大學

電機與控制工程學系

碩士論文

一種用於上行 LTE 之單載波分頻多工系統的可變長度快速傅立葉轉換處理器

A Variable FFT /IFFT Processor for SC-FDMA Systems over Uplink LTE Applications

研究生：王星雅

指導教授：蔡尚澕　教授

中華民國一百零一年五月

一種用於上行LTE之單載波分頻多工系統的可變長度快速傅立葉轉換處理器

A Variable FFT /IFFT Processor for SC-FDMA Systems over Uplink LTE Applications

研究生：王星雅　　Student：Hsing-Ya Wang

指導教授：蔡尚澕　　Advisor：Shang-Ho Tsai

國立交通大學
電機與控制工程學系
碩士論文

A Thesis
Submitted to Institute of Electrical Control Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Electrical and Control Engineering

May 2012
Hsinchu, Taiwan, Republic of China

中華民國 一百零一 年 五 月

一種用於上行 LTE 之單載波分頻多工系統的可變長度快速傅立葉轉換處理器

學生：王星雅　　　　　　　　　　　　　　　指導教授：蔡尚澕

國立交通大學電機與控制工程學系（研究所）碩士班

摘　　　要

在這篇論文，我們介紹一個可以應用於 LTE 之單載波分頻多工系統中的可變長度快速傅立葉轉換器。這個 2048/1536/1024/512/256/128-point 可變長度快速傅立葉轉換是以 radix-2、radix-3 及 radix-2^3 快速傅立葉轉換演算法，並且利用混和 radix 演算法去執行。為了在單載波分頻多工系統中利用單一個快速傅立葉轉換處理器同時處理傅立葉和反傅立葉運算，我們仔細地設計時間規畫。因此可以減少面積。為了能更完美地設計時間規劃，我們提出了每一級重新設定的控制和輸出重新排列時的早讀。我們利用 Xilinx FPGA Vertex5-XC5VLX110T-FF1136 去模擬操作在 100M 赫茲的快速傅立葉轉換處理器。

A Variable FFT /IFFT Processor for SC-FDMA Systems over Uplink LTE Applications

Student：Hsing-Ya Wang　　　　　　　　Advisors：Dr. Shang-Ho Tsai

Department of Electrical and Control Engineering
National Chiao Tung University

ABSTRACT

In this thesis, we present a variable FFT that can support multiple FFT/IFFT size for LTE SC-FDMA systems. The 2048/1024/512/128-point variable FFT is based on radix-2, radix-3 and radix-2^3 FFT algorithm, and we use mixed-radix algorithm to perform them. To handle both FFT and IFFT operations required in the SC-FDMA systems using only one FFT/IFFT processor, we carefully schedule the timing plan. As a result, the required area can be reduced. For the architecture, we propose a ripple-like ON/OFF stage control and an early access of reordering to implement our proposed timing plan. The proposed variable FFT processor is implemented using Xilinx FPGA Vertex5-XC5VLX110T-FF1136 at 100MHz operation frequency.

誌　　謝

最先要感謝的是我的指導教授蔡尚澕教授，在我眼中教授是通訊設計領域的專業者同時是不吝於給學生任何幫助的老師，除了指導我的專業研究，在我生活遇到瓶頸時能適時的給我幫助和鼓勵，讓我最終能完成學業。老師不辭辛苦的帶領我們，也希望老師在忙碌之餘能多愛惜自己的身體。也感謝我的口試委員:林源倍教授和簡鳳村教授的指導使我的論文更加完整。

另外，感謝 815 實驗室的同學，課業上的幫助及意見的提供讓我在修課上的疑惑能夠有很大的幫助。各個同學研究領域的專業知識讓我的研究能適時得到靈感，也在研究生活中創造許多美好的回憶。

最後，我要感謝的是我偉大的母親，從小她就一直認真地栽培我，當我遇到困難她會給我滿滿的關心與安慰，是我做所有事情的動力，感謝她在背後的支持，是她成就了現在的我。

將此篇論文獻給所有關心我幫助我的人，感謝你們。

A Variable FFT/IFFT Processor for SC-FDMA Systems over Uplink LTE Applications

Hsing-Ya Wang

Advisor: Dr. Shang-Ho Tsai
Department of Electrical and Control Engineering
National Chiao Tung University

May 3, 2012

Abstract

In this thesis, we present a variable FFT that can support multiple FFT/IFFT size for LTE SC-FDMA systems. The 2048/1024/512/128-point variable FFT is based on radix-2, radix-3 and radix-2^3 FFT algorithm, and we use mixed-radix algorithm to perform them. To handle both FFT and IFFT operations required in the SC-FDMA systems using only one FFT/IFFT processor, we carefully schedule the timing plan. As a result, the required area can be reduced. For the architecture, we propose a ripple-like ON/OFF stage control and an early access of reordering to implement our proposed timing plan. The proposed variable FFT processor is implemented using Xilinx FPGA Vertex5-XC5VLX110T-FF1136 at 100MHz operation frequency.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and goal

LTE (Long Term Evolution) is a standard for wireless communication of high-speed mobile phones and data terminals, which is an evolution of the GSM/UMTS standards. It is a next generation mobile system from 3GPP (3rd Generation Partnership Project) with a focus on wireless broadband. The goal of LTE is to increase the capacity and speed of wireless data networks using new DSP (Digital Signal Processing) techniques and modulations. Downlink and uplink multiple access of LTE use OFDMA (Orthogonal Frequency-Division Multiple Access) and SC-FDMA (Single-carrier FDMA) respectively. SC-FDMA is particularly to reduce power consumption. LTE, Wi-MAX and UMB are called 4G.LTE combines OFDMA/SC-FDMA/MIMO/HARQ. Hence it has good data rate. Some important features of LTE in physical layer as follow:

- **OFDMA techniques:**
  OFDMA is used in many wireless communication standards, which is also applied to LTE downlink. In OFDM-based systems, FFT is key component and hence it is widely studied in recent years.

- **SC-FDMA techniques:**
  SC-FDMA is similar to OFDMA because they both use FFT, but SC-FDMA has lower PAPR (peak-to- average power ratio). Lower PAPR is a

great benefit to transmit signal on mobile terminal, because it can reduce power consumption.

- **Variable bandwidth**
  LTE can support variable bandwidth in physical layer. That is, it can adjust the transmission rate via changing the bandwidth. As a result, we need FFT with various sizes. For example, the bandwidth for LTE systems can be 1.4MHz, 3MHz, 5MHz, 10MHz, 15MHz, 20MHz corresponding to the 128-, 256-, 512-, 1024 -, 1536-, and 2048-point FFT. This dynamic adjustment of bandwidth and possible FFT size enables user roaming in different network. Thus, the variable FFT is a key component for OFDM systems with various bandwidth.

SC-FDMA is a new multiple access technique that uses single carrier modulation, DFT-spread orthogonal frequency multiplexing, and frequency equalization.[7] It has a similar structure and performance as OFDM, except the addition of a new DFT block. Because of that, it increases area, cost and power consumption, which is a challenging to solve the problems.

1.2 Contributions and Features

In this thesis, we propose an FFT/IFFT architecture for SC-FDMA, and implemented it using a Xilinx FPGA design board with Vertex5-XC5VLX110T-FF1136. By using time sharing technique, the proposed FFT/IFFT can handle both the FFT and IFFT in SC-FDMA using one FFT processor. Hence, it reduces the hardware complexity. In this thesis, we will show how to perform time sharing. Moreover, the used FPGA design flow will also be described and demonstrated. Detailed architecture specifications of the proposed FFT/IFFT processor are listed as follows:

- High radix to reduce the complexity: Because higher-radix algorithm can reduce number of multiplier and power consumption, we employ radix-2^3 algorithm to implement the last two stages.

- Subcarrier mapping: By our design, we can do subcarrier mapping in reordering step. We do not need unnecessary latency to perform that.

- Make FFT/IFFT work at the same time: SC-FDMA system has both FFT and IFFT, because FFT has similar structure with IFFT, we use timing plan to do FFT and IFFT at the same time. Thus, we almost can reduce a half of the hardware.

- Early access of reordering: By our calculation, we can read the FFT output data earlier by using ripple-like ON/OFF stage control. Thus, timing plan can be designed better.

Chapter 2

Background

2.1 FFT/IFFT for SC-FDMA vs OFDM

SC-FDMA (Single-carrier Frequency-Division Multiple Access) is a technique for high data uplink communication and has been adopted by LTE. Fig. 2.1 shows a block diagram of a SC-FDMA system. SC-FDMA has similar throughput performance and complexity with OFDM, while its PAPR (peak-to-average power ratio) is much smaller than OFDM systems, which makes it suitable for uplink transmission. SC-FDMA is often viewed as FFT coded OFDM. The use of FFT transforms the time-domain symbols to the frequency-domain before passing through the standard OFDM modulation. Thus, SC-FDMA has all the advantages of OFDM. The main advantage of OFDM, as is for SC-FDMA, is its robustness against multipath signal propagation, which is suitable for broadband systems.

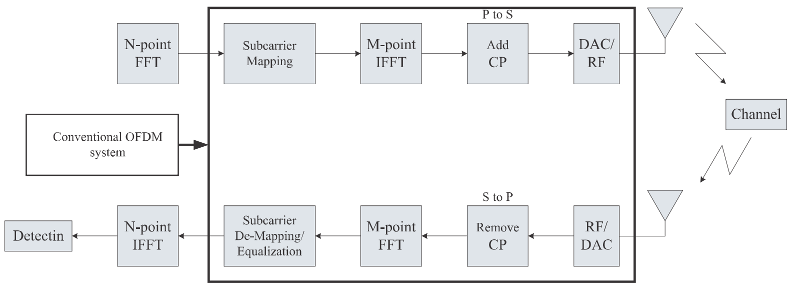


Figure 2.1: SC-FDMA Transmitter and Receiver.

2.2 FFT algorithm

FFT algorithm has several types, including radix-2, radix-4, radix-8, radix-2^2 and radix-2^3. Radix-2 is the most simple and elastic structure of all FFT algorithms. In many researches, we know higher radix has lower operations which results in good efficiency, but it increases complexities. Radix-2^2 and radix-2^3 have simple structure of radix-2 and lower operations of higher radix. Therefore, we use series connection to combine radix-2 and radix-2^3. Because LTE standard has 1536-point FFT, we use radix-3 for implementation. FFT has two mathematics modes which are DIT (decimation in time) and DIF (decimation in frequency). We use DIF mode in follows reduction.

2.2.1 Radix-2 algorithm

A basic N-point discrete Fourier transform (DFT) is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn}, \quad k = 0, 1, \ldots, N-1, \tag{2.1}$$

where $x(n)$ and $X(k)$ are complex number. The twiddle factor is

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}} = \cos\frac{2\pi nk}{N} - j\sin\frac{2\pi nk}{N} \tag{2.2}$$

Divide $x(n)$ into even part and odd part

$$\begin{aligned}
X(2r) &= \sum_{n=0}^{N-1} x(n)W_N^{2rn}, \quad r = 0, 1, \ldots, N/2-1 \\
&= \sum_{n=0}^{N/2-1} x(n)W_N^{2rn} + \sum_{n=N/2}^{N-1} x(n)W_N^{2rn} \\
&= \sum_{n=0}^{N/2-1} x(n)W_N^{2rn} + \sum_{n=0}^{N/2-1} x(n+N/2)W_N^{2r(n+N/2)} \\
&= \sum_{n=0}^{N/2-1} [x(n)+x(n+N/2)]W_N^{2rn} \quad (W_N^{rn} = 1) \\
&= \sum_{n=0}^{N/2-1} [x(n)+x(n+N/2)]W_{N/2}^{rn} \quad (W_N^{2rn} = W_{N/2}^{rn})
\end{aligned} \tag{2.3}$$

$$\begin{aligned} X(2r+1) &= \sum_{n=0}^{N-1} x(n)W_N^{(2r+1)n}, \quad r = 0, 1, \ldots, N/2-1 \\ &= \sum_{n=0}^{N/2-1} x(n)W_N^{(2r+1)n} + \sum_{n=N/2}^{N-1} x(n)W_N^{(2r+1)n} \\ &= \sum_{n=0}^{N/2-1} x(n)W_N^{(2r+1)n} + \sum_{n=0}^{N/2-1} x(n+N/2)W_N^{(2r+1)(n+N/2)} \\ &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_N^{(2r+1)n} \quad (W_N^{rN+N/2} = -1) \\ &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_N^{n}W_N^{2rn} \\ &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_N^{n}W_{N/2}^{rn} \end{aligned} \tag{2.4}$$

Take N=8 for instance, x[0]-x[3] and x[4]-x[7] add and subtract mutually as shown in Fig 2.2, then the subtracted values are multiplied by twiddle factors.

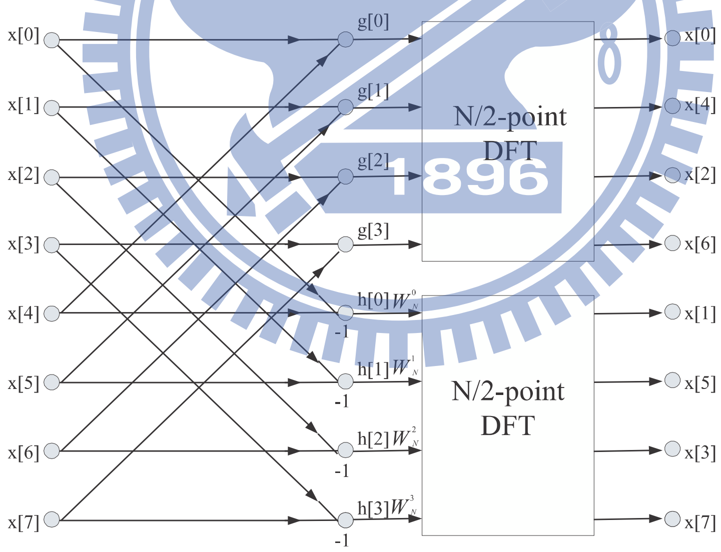


Figure 2.2: Simple DIF 8-point FFT

Use the same axiom to go on decomposing, shown in Fig 2.3. After several times of decomposing, we can obtain Fig 2.4, a complete DIF 8-point FFT with radix-2.

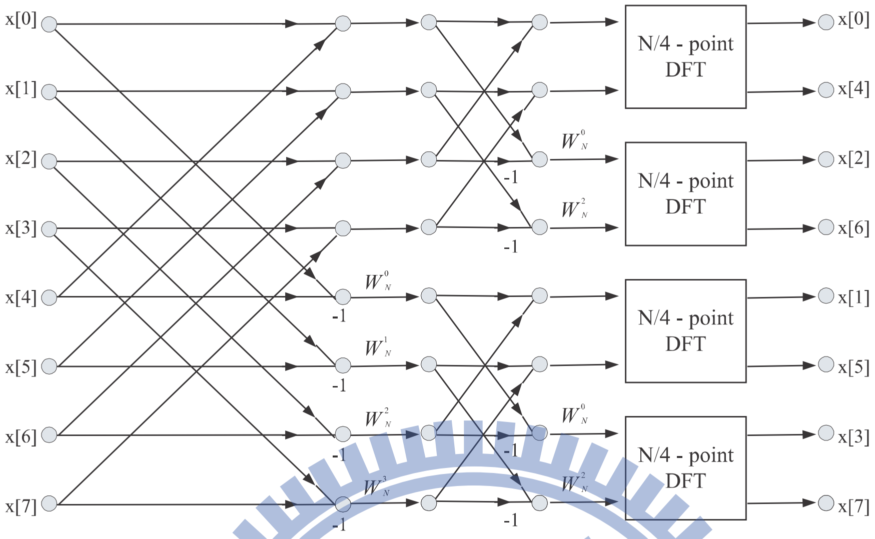


Figure 2.3: Simple DIF 8-point FFT

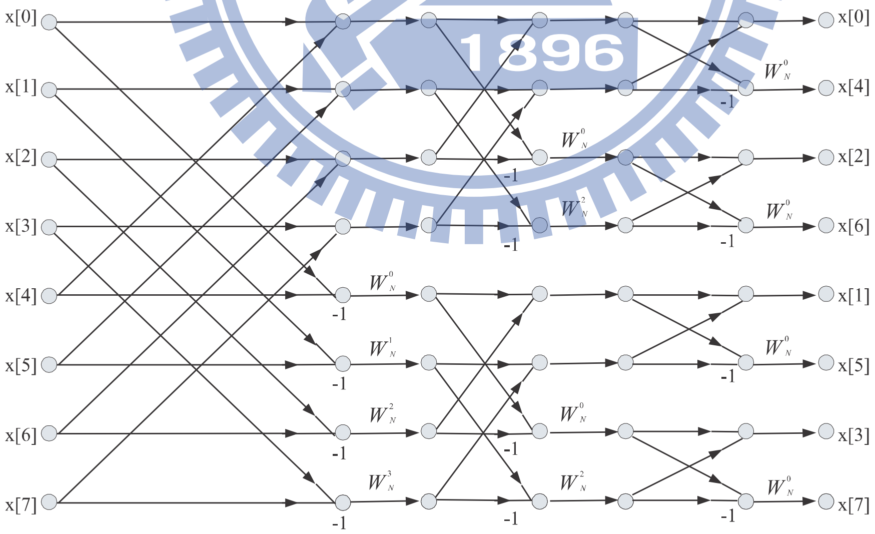


Figure 2.4: DIF 8-point FFT with radix-2

2.2.2 Radix-2^3 algorithm

8^N-point (N is an integer) FFT usually uses radix-8 or radix-2^3 algorithm for implemention. They have advantage in trivial twiddle factor, but radix-8 structure demands more complexities than radix-2^3. Hence we usually use radix-2^3 instead of radix-8 in implemention. Radix-2^3 algorithm can be written as follows.

$$write \quad k \quad = l_1 + 2l_2 + 4l_3 + 8k_1 \tag{2.5}$$

$$\begin{aligned}
& X(8k_1 + 4l_3 + 2l_2 + l_1) \\
= \; & \sum_{n=0}^{N/8-1} \{[x(n) + x(n + \frac{2N}{8}) + W_4^{l_1} x(n + \frac{4N}{8}) + W_4^{2l_1} + x(n + \frac{6N}{8})W_4^{-1}] \\
& +[x(n + \frac{N}{8}) + x(n + \frac{3N}{8}) + W_4^{l_1} x(n + \frac{5N}{8}) + W_4^{2l_1} x(n + \frac{7N}{8})W_4^{-1}]W_8^{-1} \\
& \}W_N^{nl} W_{N/8}^{nk_1} \\
= \; & \sum_{n=0}^{N/8-1} \{[x(n) + W_2^{l_1} x(n + \frac{4N}{8}) + W_2^{l_2} W_4^{l_1} x(n + \frac{2N}{8}) + W_2^{l_1} x(n + \frac{6N}{8})] \\
& +[x(n + \frac{N}{8}) + W_2^{l_1} x(n + \frac{5N}{8}) + W_2^{l_2} W_4^{l_1} x(n + \frac{3N}{8}) + W_2^{l_1} x(n + \frac{7N}{8})]W_8^{2l_2+l_1} \\
& \}W_N^{n(4l_3+2l_2+l_1)} W_{N/8}^{nk_1}
\end{aligned}$$

$$l_1, l_2, l_3 = 0, 1 \qquad k = 0, 1, \ldots, N/8 - 1 \tag{2.6}$$

Fig 2.5 shows DIF butterfly with radix-2^3. From Fig 2.6, we can see that radix-2^3 has the simple basic stucture of radix-2, but the twiddle factor was replaced by trivial twiddle factor (W_4^N, W_8^N). Thus we say radix-2^3 algorithm is good for implemention.

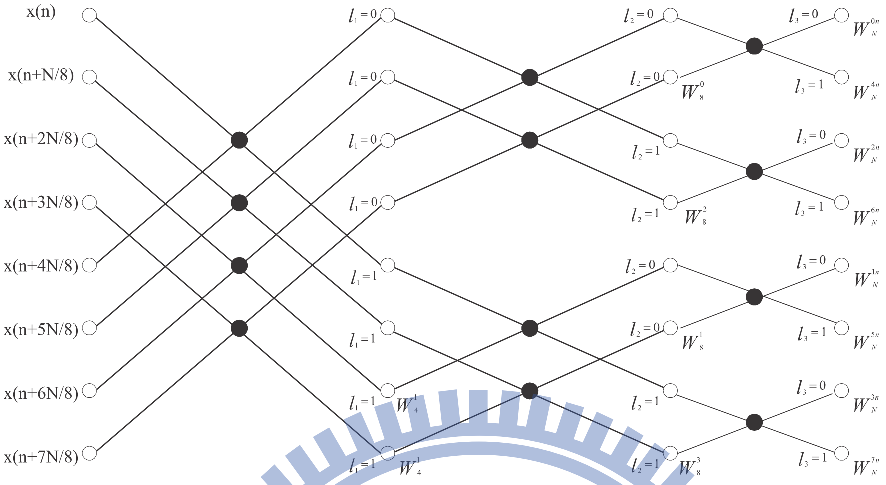


Figure 2.5: DIF butterfly with radix-2^3

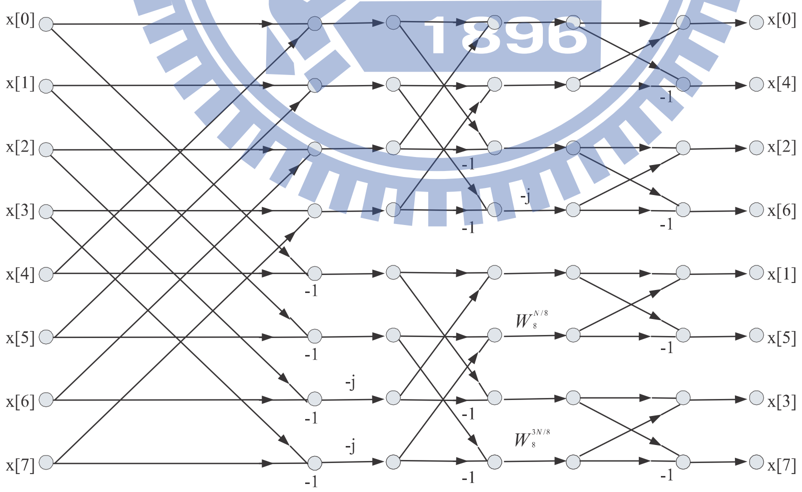


Figure 2.6: DIF 8-point FFT with radix-2^3

2.2.3 Radix-3 algorithm

$$X(3k_1+l) \;=\; \sum_{n=0}^{N/3-1}[X(n)+x(n+\frac{N}{3})W_3^l+x(n+\frac{2N}{3})W_3^{2l}]W_N^{nl}W_{N/3}^{nk_1}$$

$$l=0,1,2 \qquad k=0,1,\ldots,N/3-1 \tag{2.7}$$

Fig. 2.7 shows a radix-3 butterfly.

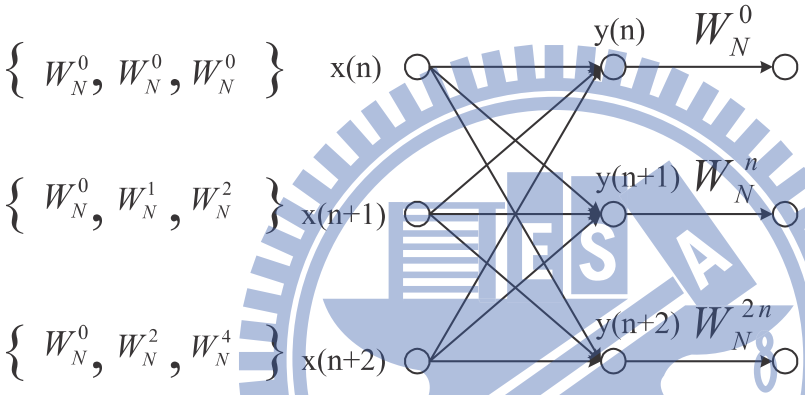

$\{\ W_N^0, W_N^0, W_N^0\ \}$
$\{\ W_N^0, W_N^1, W_N^2\ \}$
$\{\ W_N^0, W_N^2, W_N^4\ \}$
x(n)
x(n+1)
x(n+2)
y(n)
y(n+1)
y(n+2)
W_N^0
W_N^n
W_N^{2n}

Figure 2.7: DIF butterfly with radix-3

From 2.7 , the input should be multiplied by extra twiddle factors and then passed to the butterfly. More specifically,

$$\begin{aligned} y(n) &= x(n)W_N^0 + x(n+1)W_N^0 + x(n+2)W_N^0 \\ y(n+1) &= x(n)W_N^0 + x(n+1)W_N^1 + x(n+2)W_N^2 \\ y(n+2) &= x(n)W_N^0 + x(n+1)W_N^2 + x(n+2)W_N^4 \end{aligned}$$

An example for a 9-point FFT butterfly with radix-3 is shown in 2.8.

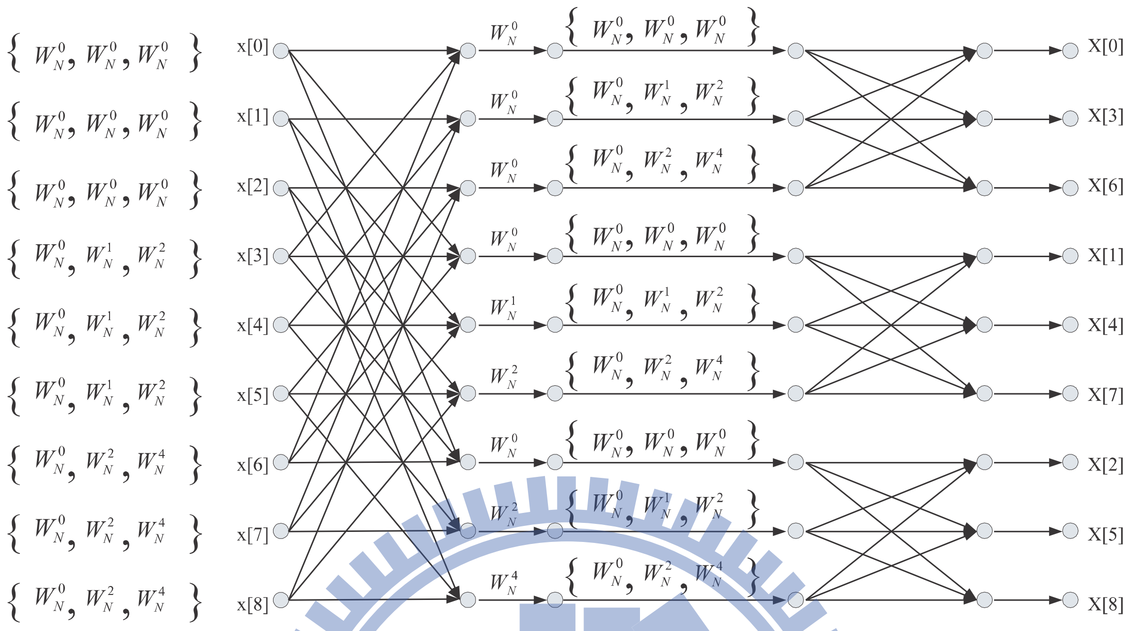


Figure 2.8: DIF 9-point FFT with radix-3

2.2.4 FFT/IFFT

The IFFT of an N-point sequence $x(n)$ is defined as

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk} \qquad k = 0, 1, \ldots, N-1 \tag{2.8}$$

In order to implement the IFFT algorithm more efficiently, equation (2.9) can be rewritten as

$$x(n) = \frac{1}{N}\left\{\sum_{k=0}^{N-1} X^*(k)W_N^{nk}\right\}^* \tag{2.9}$$

According to equation (2.10), the IFFT can be performed by adding the complex conjugate of input data and output data without changing any coefficient in the original FFT architecture. We can see Fig. 2.9, it utilizes multiplexer to change the mode that operation of FFT or IFFT. Thus, the hardware implementation can be more efficient. The data sequence is divided into two data paths by commutator, and then properly add and substract mutually for two data paths.

The PE (processor element) is implemented by radix-2 algorithm. The numbers of PE unit, multipliers and delay elements are with order $log_2 N$ $(log_2 N - 2)$ and $(\frac{3N}{2}) - 2$ respectively.

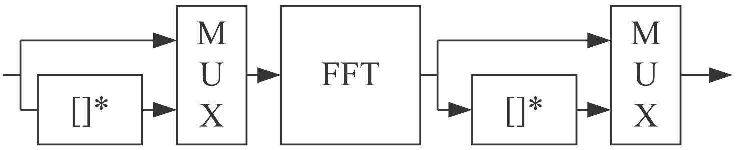


Figure 2.9: Block diagram of FFT/IFFT.

2.3 Variable FFT

Many standars need various FFT, such as DAB, DVB-T, VDSL, Wi-MAX and LTE. Their FFT sizes are shown in Table 2.1. Hence, the design of a scalable FFT for different purposes become more important. For variable FFT, we use mixed-radix algorithm which is more elastic and convenient for variable FFT sizes. We utilize the above radix-2,radix-2^3 and radix-3 for mixed-radix to present a variable FFT which can support 64, 32, 16 and 8-point operation. It is very easy to modify our LTE standar designed by 128, 256, 512, 1024, 1536 and 2048-point FFT operation size to create any required length of FFT. Hence, this modification of circuit is simple and convenient to be used for many FFT sizes of other standards. For the others scalable FFT, please refer to [2].

Communication system	FFT size
LTE	128,256,512,1024,1536,2048
Wi-MAX [6]	128,512,1024,2048
VDSL [3]	8192,4096,2048,1024,512
DAB [4]	2048,1024,512,256
DVB-T [5]	8192,2048

Table 2.1: FFT size in several OFDM or SC-FDMA systems

2.3.1 Pipeline FFT processor architecture

The traditional radix-2 pipeline FFT architectures can be roughly classified into MDC (multi path delay commutator) and SDF (single path delay feedback) [1]. Fig. 2.10 shows a radix-2 multi-path delay commutator (R2MDC) architecture with N=8. The data sequence is divided into two data paths by commutator, and then properly add and substract mutually for two data paths. The PE (processor element) is implemented by radix-2 algorithm. The numbers of PE unit, multipliers and delay elements are with order log_2N (log_2N-2) and $(\frac{3N}{2})-2$ respectively. Fig. 2.11 shows a radix-2 single path delay feedback (R2SDF) architecture with N=8. The use of delay elements in R2SDF is more efficient than R2MDC by sharing the memory. The numbers of PE units, multipliers and delay elements for R2MDC are $N-1$, (log_2N-1) and log_2N. The SDF FFT and MDC FFT are described as follows.

- a) MDC FFT: Because the MDC FFT utilizes multi path to increase data path, the MDC FFT can increase the throughput rate. Its drawback is that the memory size become much more.

- b) SDF FFT: Because the SDF FFT utilizes feedback to reuse memory, the SDF FFT can reduce the memory usage. Its drawback is that the throughput rate is low.

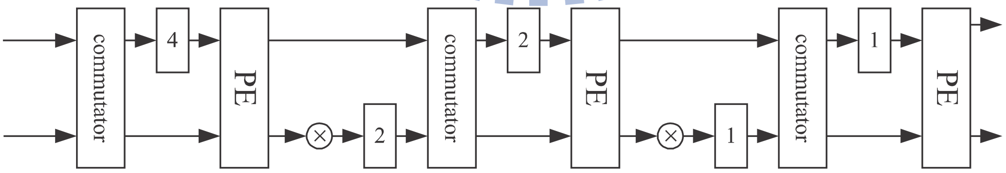


Figure 2.10: Architecture of R2MDC.

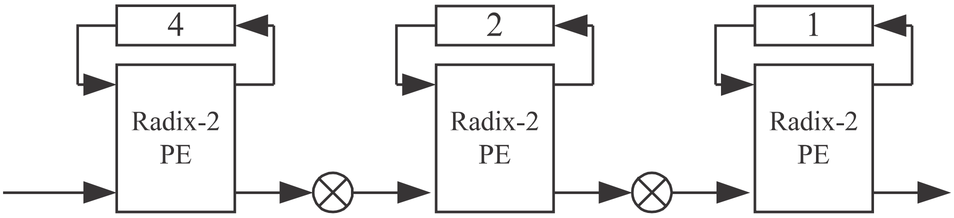


Figure 2.11: Architecture of R2SDF.

2.3.2 Variable FFT processor architecture

We can see a variable FFT which can achieve 64-point, 32-point, 16-point and 8-point operation in Fig. 2.12. The structure uses mixed-radix algorithm for implemention, where stage 1 to stage 3 are radix-2 and stage 4 is radix-2^3. It utilizes above SDF structure and multiplexors to perform the variable FFT. There are four different FFT lengths. For the 64-point FFT, all stages are active. For the 32-point FFT, the input data will skip the first stage and go to the second stage directly. For the 16-point FFT, the input data will skip the first stage and the second stage and go to the third stage. Multiplexors are used to control different FFT size operation. Thus, we can use that to perform variable FFT easily.

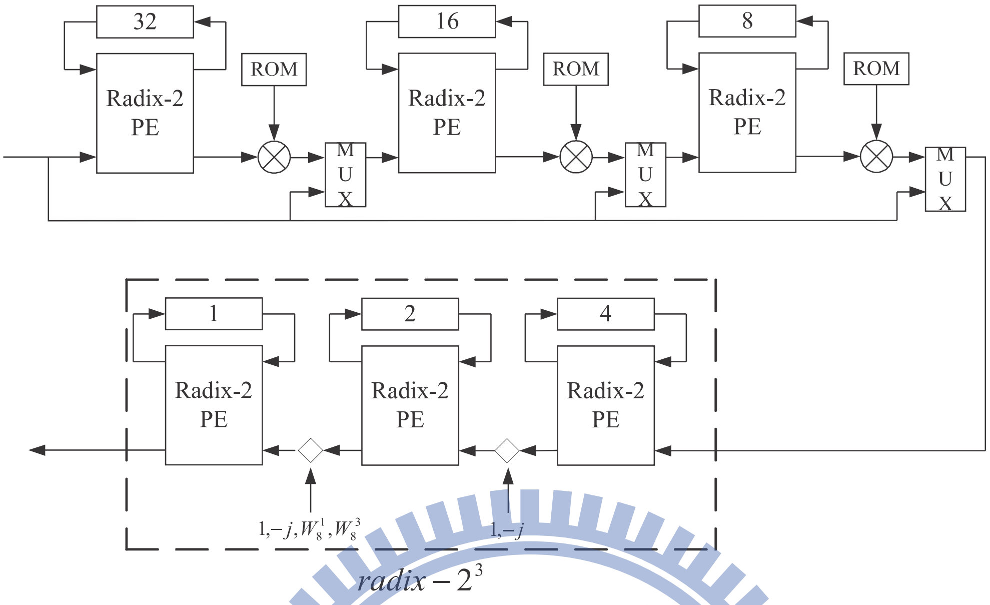


Figure 2.12: Block diagram of a variable FFT for 64-point, 32-point, 16-point and 8-point operation

2.3.3 Subcarrier Mapping

SC-FDMA is a promising technique for high data rate uplink communications. It is a modified form of OFDMA, and has similar throughput performance with OFDMA. A principle advantage is the peak-to-average power (PAPR), which is lower than OFDMA. In the SC-FDMA system structure, subcarrier mapping is a important factor which affects PAPR and throughput performance. There are many accurate analyses of throughput and PAPR with subcarrier mapping in SC-FDMA [8]. There are two ways to apportion subcarriers among terminals, which are localized SC-FDMA (LFDMA) and distributed SC-FDMA. One realization of distributed SC-FDMA is interleaved FDMA (IFDMA). The LFDMA and IFDMA are described as follows.

- a) LFDMA (localized SC-FDMA): Each terminal utilizes a set of adjacent subcarriers to transmit its symbols. Thus, the bandwidth of LFDMA is limited to a fraction of all the system bandwidth. Fig. 2.10 (a)

- b) IFDMA (distributed SC-FDMA): The subcarriers used by a terminal are spread over the entire signal band. The occupied subcarriers of users are allocated uniformly as shown in Fig. 2.10 (b).

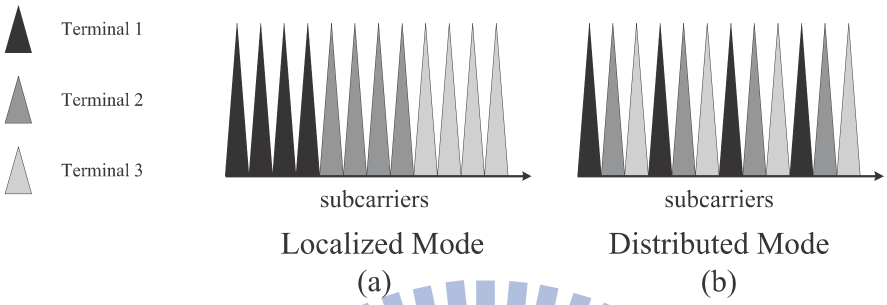


Figure 2.13: Subcarrier allocation methods for multiple (3 users, 12 subcarriers, and 4 subcarriers per user)

An example of subcarrier mapping is in Fig. 2.14. We assume that 12 subcarriers are shared by 3 users, so each user has four data symbols to transmit at the same time. The DFT output has four complex data symbols in frequency domain, and the four data symbols are mapped on 12 subcarriers using different mapping methods.

In [8], the authors analyzed the effects of subcarrier mapping on throughput and PAPR. They find LFDMA with channel-dependent scheduling (CDS) has higher throughput than IFDMA, but the PAPR performance of IFDMA is better than LFDMA.

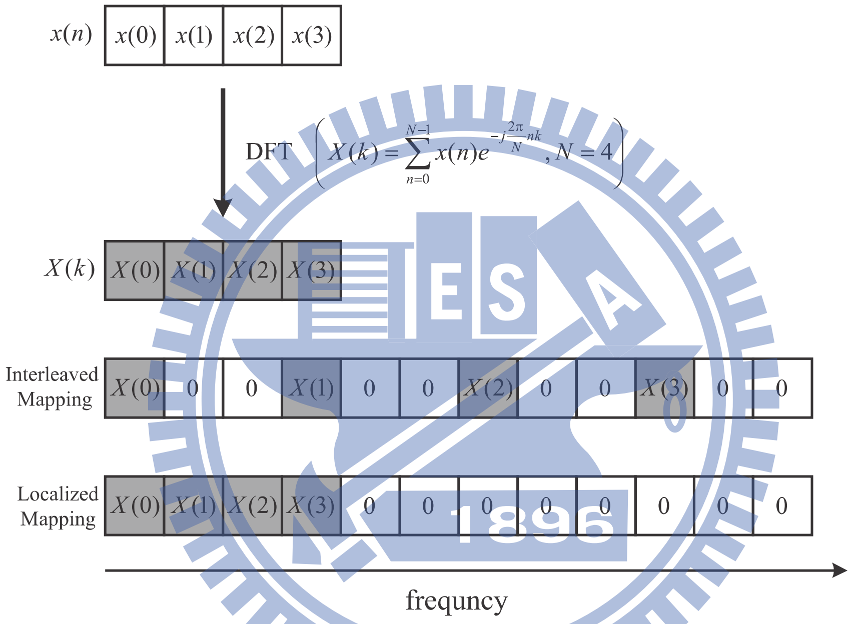


Figure 2.14: Subcarrier mapping example in the frequency domain for 4 subcarriers per user, 3 users, and 12 subcarriers

Chapter 3

The proposed scalable LTE for LTE systems

3.1 Algorithm

From (2.1), the N-point DFT operation can be decomposed to $N_1 \times N_2 \times \ldots \times N_k$ point DFT operation. We use the radix-2^3 as many as possible reduce the multipliers. The mathematical representation is shown in equation (3.1).

$$N = 2048 = \underbrace{\underbrace{\underbrace{\underbrace{\underbrace{8 \times 8 \times 2}_{\text{128-point}} \times 2}_{\text{256-point}} \times 2}_{\text{512-point}} \times 2}_{\text{1024-point}} \times 2}_{\text{2048-point}} \tag{3.1}$$

From (3.1), $N = 2048 = N_1 \times N_2 = 2 \times 1024$. Define the following indices

$$n = N_2 n_1 + n_2 = 1024 n_1 + n_2, \qquad \left\{ \begin{array}{l} n_1 = 0, 1 \\ n_2 = 0, 1, \ldots, 1023 \end{array} \right. ,$$

and

$$k = k_1 + N_1 k_2 = k_1 + 2k_2, \qquad \left\{ \begin{array}{l} k_1 = 0, 1 \\ k_2 = 0, 1, \ldots, 1023 \end{array} \right.$$

(2.1) can be rewritten as

$$X(k_1 + 2k_2)$$

$$
\begin{aligned}
&= \sum_{n_2=0}^{1023} \sum_{n_1=0}^{1} x(1024n_1+n_2)W_{2048}^{(1024n_1+n_2)(k_1+2k_2)} \\
&= \underbrace{\sum_{n_2=0}^{1023} \left\{ \underbrace{\sum_{n_1=0}^{1} x(1024n_1+n_2)W_2^{n_1k_1}}_{\text{2-point}} \quad \underbrace{W_{2048}^{n_2k_1}}_{\text{twiddle factor}} \right\} W_{1024}^{n_2k_2}}_{\text{1024-point}} \qquad (3.2) \\
&= \sum_{n_2=0}^{1024} B_2^{(1)}(n_2)W_{1024}^{n_2k_2}, \qquad (3.3)
\end{aligned}
$$

where $B_r^{(k)}$ denotes the radix-r algorithm at stage k.

Now $N_2 = 1024 = N_2 \times N_3 = 2 \times 512$. Define the indices n_2 and k_2 as

$$
n_2 = N_3n_2 + n_3 = 512n_2 + n_3, \qquad \begin{cases} n_2 = 0, 1 \\ n_3 = 0, 1, ...511 \end{cases},
$$

and

$$
k_2 = k_2 + N_2k_3 = k_2 + 2k_3, \qquad \begin{cases} k_2 = 0, 1 \\ k_3 = 0, 1, ..., 511 \end{cases}
$$

We have:

$$
\begin{aligned}
& X(k_1 + 2k_2 + 4k_3) \\
&= \sum_{n_3=0}^{511} \sum_{n_2=0}^{1} B_2^{(1)}(512n_2+n_3)W_{1024}^{(512n_2+n_3)(k_2+2k_3)} \\
&= \underbrace{\sum_{n_3=0}^{511} \left\{ \underbrace{\sum_{n_2=0}^{1} B_2^{(1)}(512n_2+n_3)W_2^{n_2k_2}}_{\text{4-point}} \quad \underbrace{W_{1024}^{n_3k_2}}_{\text{twiddle factor}} \right\} W_{512}^{n_3k_3}}_{\text{512-point}} \\
&= \sum_{n_3=0}^{511} B_2^{(2)}(n_3)W_{512}^{n_3k_3} \qquad (3.4)
\end{aligned}
$$

In this way, we can obtain the result is given by

$$
\begin{aligned}
& X(k_1 + 2k_2 + 4k_3 + 8k_4 + 16k_5 + 32k_6) \\
&= \sum_{n_6=0}^{63} B_2^{(5)}(n_6)W_{64}^{n_6k_6}, \qquad (3.5)
\end{aligned}
$$

where $k_6 = 0, 1, \ldots, 63$. By decomposing the 64-point DFT into the 8-point DFT, we can achieve the 2048-point mixed-radix FFT algorithm.

$$\begin{aligned} & X(k_1 + 2k_2 + 4k_3 + 8k_4 + 16k_5 + 32k_6 + 256k_7) \\ = & \sum_{n_7=0}^{7} \left\{ \sum_{n_6=0}^{7} B_2^{(5)}(8n_6 + n_7) W_8^{n_6 k_6} W_{64}^{n_7 k_6} \right\} W_8^{n_7 k_7} \end{aligned} \tag{3.6}$$

Because the radix-8 butterfly unit is inefficient in the use of adders and multipliers. we use the radix-2^3 FFT algorithm [1] to replace the radix-8 FFT algorithm. In this case, we can further reduce the complexity of the butterfly by using the radix-2 butterfly three times. The SFG of the 2048-point mixed-radix FFT algorithm, is as shown in Fig. 3.1 and Fig. 3.2.

Let $n_6 = 4\alpha_1 + 2\alpha_2 + \alpha_3$ and $k_6 = \beta_1 + 2\beta_2 + 4\beta_3$, we can obtain the form with radix-2^3 in equation (3.9).

$$\begin{aligned} & X(k_1 + 2k_2 + 4k_3 + 8k_4 + 16k_5 + 32(\beta_1 + 2\beta_2 + 4\beta_3) + 256k_7) \\ = & \sum_{n_7=0}^{7} B_8^{(6)}(n_7, k_7) W_8^{n_7 k_7}, \end{aligned} \tag{3.7}$$

where

$$\begin{aligned} B_8^{(6)}(n_7, k_7) = & \sum_{\alpha_3=0}^{1} \sum_{\alpha_2=0}^{1} \sum_{\alpha_1=0}^{1} B_2^{(5)}(8(4\alpha_1 + 2\alpha_2 + \alpha_3) + n_7) W_2^{\alpha_1 \beta_1} \\ & W_4^{\alpha_2 \beta_1} W_2^{\alpha_2 \beta_2} W_8^{\alpha_3(\beta_1 + 2\beta_2)} W_2^{\alpha_3 \beta_3} W_{64}^{n_7(\beta_1 + 2\beta_2 + 4\beta_3)} \end{aligned} \tag{3.8}$$

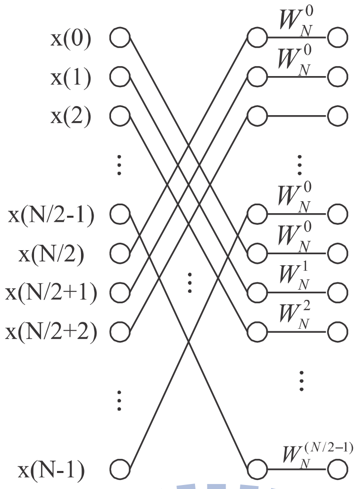


Figure 3.1: The SFG of stage 1 to stage 5 (radix-2).

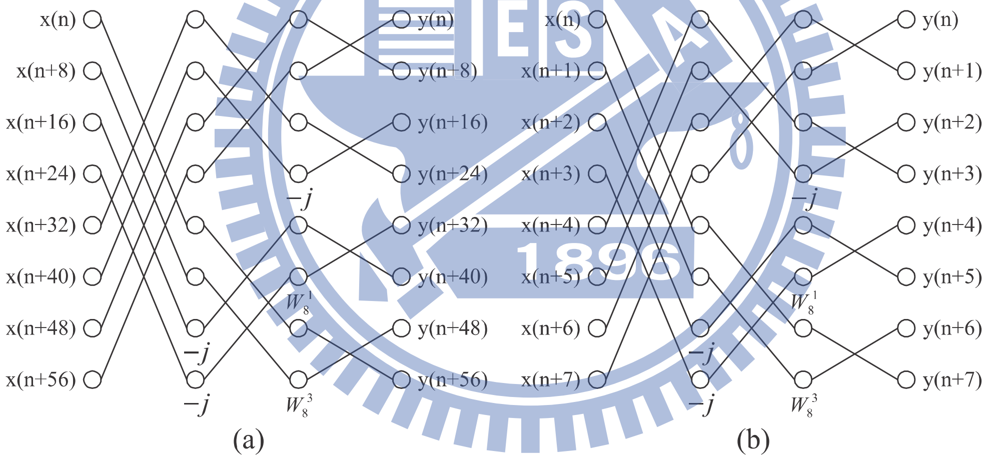


Figure 3.2: The SFG of stage 6 to stage 7 (radix-2^3).

LTE standard has 1536-point FFT, and the above equations only perform 128-point, 256-point, 512-point, 1024-point and 2048-point algorithm. Hence we derive the 1536-point algorithm as follows.

$$N = 1536 = \underbrace{\underbrace{\underbrace{\underbrace{8 \times 8 \times 2}_{\text{128-point}} \times 2}_{\text{256-point}} \times 2}_{\text{512-point}} \times 3}_{\text{1536-point}} \tag{3.9}$$

From (3.9), $N = 1536 = N_1 \times N_2 = 3 \times 512$. Define the following indices

$$n = N_2 n_1 + n_2 = 512n_1 + n_2, \qquad \begin{cases} n_1 = 0, 1, 2 \\ n_2 = 0, 1, \ldots, 511 \end{cases},$$

and

$$k = k_1 + N_1 k_2 = k_1 + 3k_2, \qquad \begin{cases} k_1 = 0, 1, 2 \\ k_2 = 0, 1, \ldots, 511 \end{cases}$$

(2.1) can be rewritten as

$$\begin{aligned} & X(k_1 + 3k_2) \\ = & \sum_{n_2=0}^{511} \sum_{n_1=0}^{1} x(512n_1 + n_2) W_{1536}^{(512n_1+n_2)(k_1+3k_2)} \\ = & \underbrace{\sum_{n_2=0}^{511} \left\{ \underbrace{\sum_{n_1=0}^{1} x(512n_1 + n_2) W_2^{n_1k_1}}_{\text{3-point}} \quad \underbrace{W_{1536}^{n_2k_1}}_{\text{twiddle factor}} \right\} W_{512}^{n_2k_2}}_{\text{512-point}} \qquad (3.10) \\ = & \sum_{n_2=0}^{511} B_3^{(1)}(n_2) W_{512}^{n_2k_2}, \qquad (3.11) \end{aligned}$$

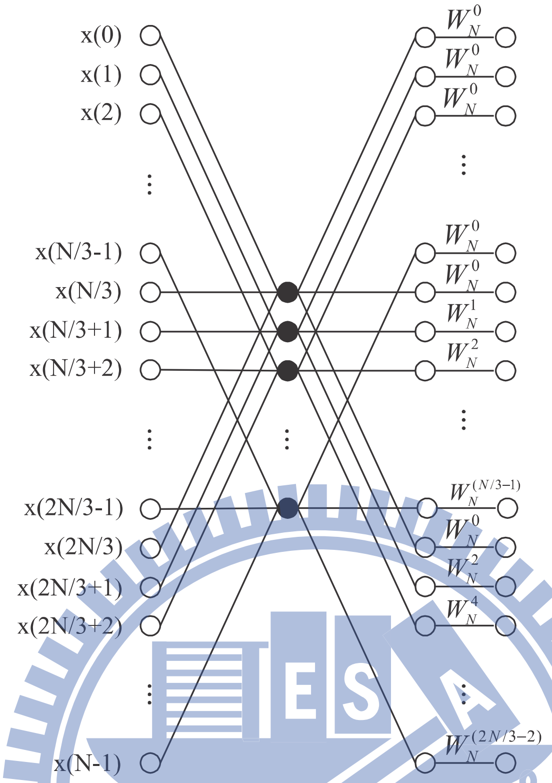


Figure 3.3: The SFG of DIF with radix-3

3.2 Architecture

The proposed architecture of the variable FFT/IFFT processor is shown in Fig. 3.4. In uplink transmission, SC-FDMA needs both FFT and IFFT. Thus we share FFT and IFFT hardware to reduce area and power. For FFT/IFFT sharing, we plan a suitable timing achedule, so that the FFT/IFFT is operated at 100MHZ and can handle a 25MHZ input throughput rate. We use LFSR (Linear Feedback Shift Register) to create variable data rate for simulation convenience. The variable FFT for LTE system provide 2048/1536/1024/512/256/128-point FFT/IFFT operations, and we design subcarrier mapper to map N point FFT to M point IFFT. After the FFT/IFFT, we use RAM to bitreverse the FFT/IFFT.

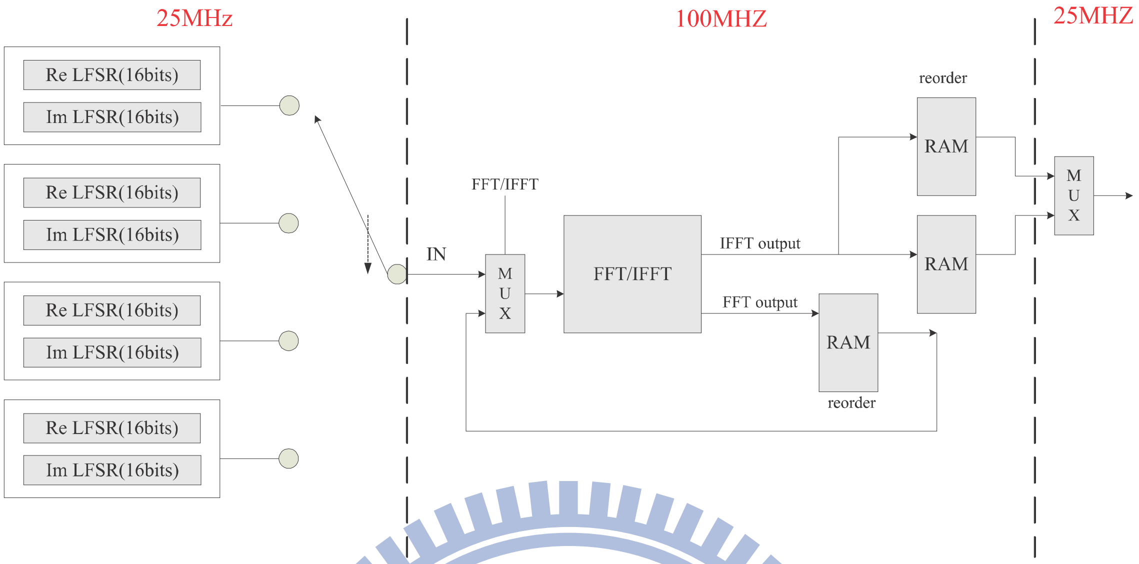


Figure 3.4: Block diagram of the variable FFT processor in FPGA.

3.2.1 FFT Processor

Fig. 3.5 shows a variable FFT processor for 128/256/512/1024/1536 and 2048-point. From subsection 3.2.2, we can skip some stages to achieve variable length FFT.

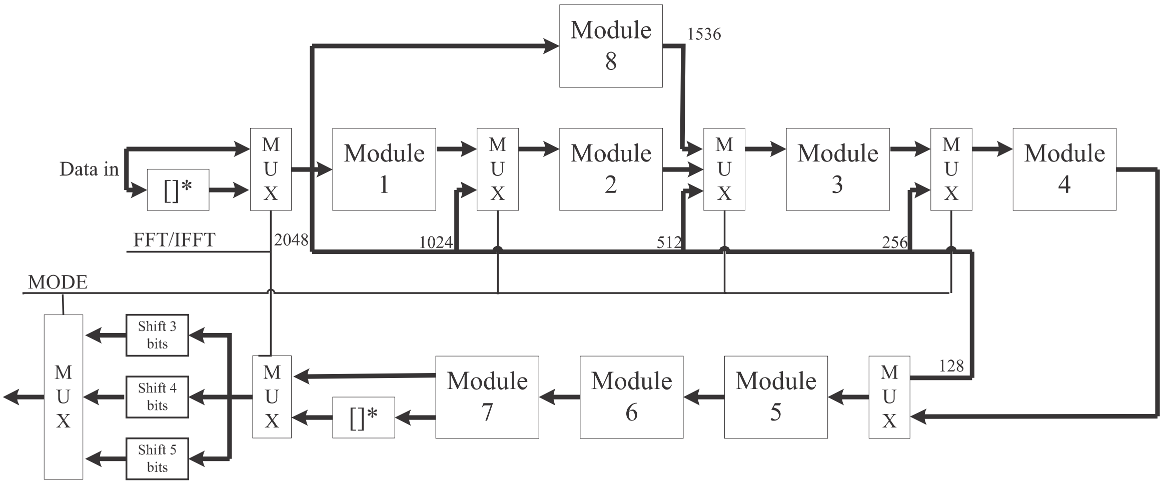


Figure 3.5: Block diagram of the variable FFT processor.

- **Module 1 to Module 5 (radix-2 FFT algorithm)**

  Module 1 to module 5 contain one butterfly unit, one multiplier, ROM table and one memory bank per module, which is shown in Fig. 3.6. Their memory bank sizes are with order 1024, 512, 256, 128 and 64 respectively.

- **Module 6 and Module 7 (radix-2^3 FFT algorithm)**

  Module 6 and module 7 contain three butterfly units and three memory bank per module, which is shown in Fig. 3.7. The size of the memory bank from stage one to stage three of module 6 are with order 32, 16 and 8 respectively. The size of the memory bank from stage one to stage three of module 7 are with order 4, 2 and 1 respectively. The multiplier of trivial twiddle factors such as w_8^N and w_4^N can be replaced by shift adders.

- **Module (radix-3 FFT algorithm)**

  Module 8 contains one butterfly unit, one multiplier, ROM table and two memory banks, which is shown in Fig. 3.8. We can see that the butterfly unit is different from other modules, i.e. module 8 is radix-3 FFT. The input data have to be multiplied by W_N^1, W_N^2 and W_N^4 first.

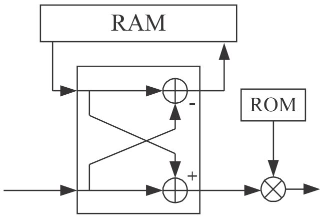


Figure 3.6: Block diagram of module 1 module 5 (radix-2)

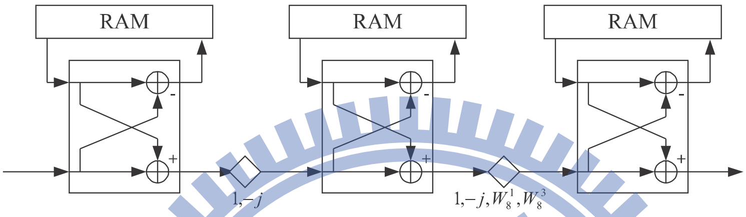


Figure 3.7: Block diagram of module 6 and module 7 (radix-2^3)

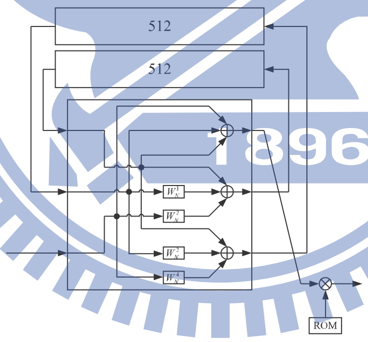


Figure 3.8: Block diagram of module 8 (radix-3)

3.2.2 Subcarrier mapper

In SC-FDMA uplink system, N-point FFT is mapped to M-point IFFT. We can operate that in reordering step, which is shown in Fig.3.9.

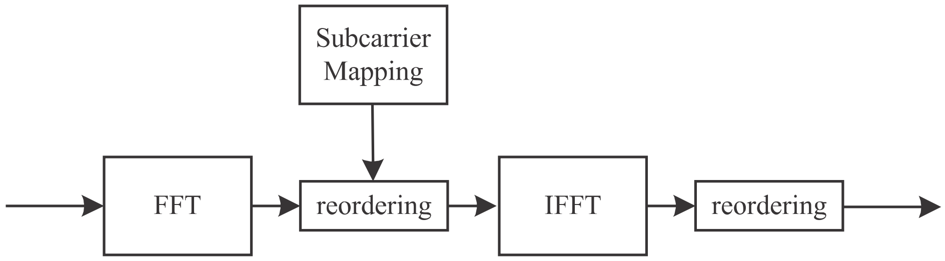


Figure 3.9: Block diagram of FFT and IFFT with subcarrier mapper

- **Localized subcarrier mapping**
  We save the index of X in the ROM. Then we read the adresses from the ROM to write the FFT output data into RAM, which is shown in Fig. 3.10 with 4-point FFT output mapping to the input of 8-point IFFT.

- **Distrubuted subcarrier mapping**
  Distrubuted subcarrier mapping has the same access control with localized subcarrier mapping. The adresses of the distrubuted subcarrier mapping can be obtained from some mathematical formula described below:
  When $N = 2^x$-point FFT is mapped to $M = 2^y$-point FFT, we should add $y - x$ bits zero, then bitreverse all bits of the FFT. Take an example for N=4 and M=8 in Fig. 3.11, we add one zero and then bitreverse the three bits. Because 1536-point is mapped to 2048-point which is particular, we find another method to implement that. First we divide the index by 3 and then do floor function. Finally bitreverse all bits of the FFT as above. Fig. 3.12 is an example for N=6 and M=8.

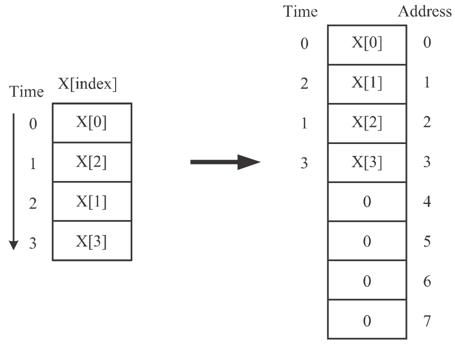
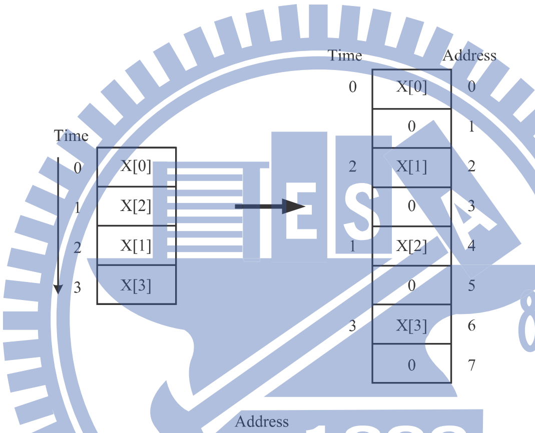
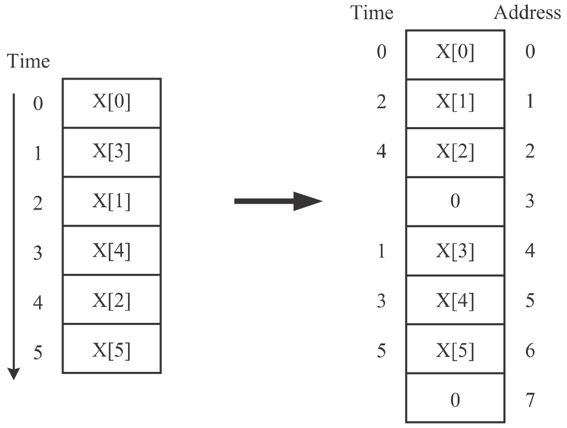


Figure 3.10: Localized subcarrier mapper with N=4 mapping to M=8



X[index]	Address (index, n bits zero)	New address
X[0]	{ 0 0 , 0 }	0
X[2]	{ 1 0 , 0 }	4
X[1]	{ 0 1 , 0 }	2
X[3]	{ 1 1 , 0 }	6

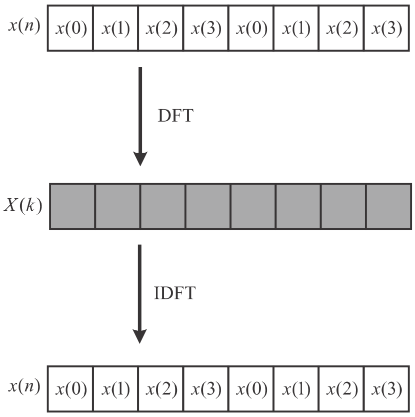
Figure 3.11: Distrubuted subcarrier mapper with N=4 mapping to M=8



X[index]	Original address	⌊Address/3⌋	New address = original address + ⌊address/3⌋
X[0]	0	0	0
X[3]	3	1	4
X[1]	1	0	1
X[4]	4	1	5
X[2]	2	0	2
X[5]	5	1	6

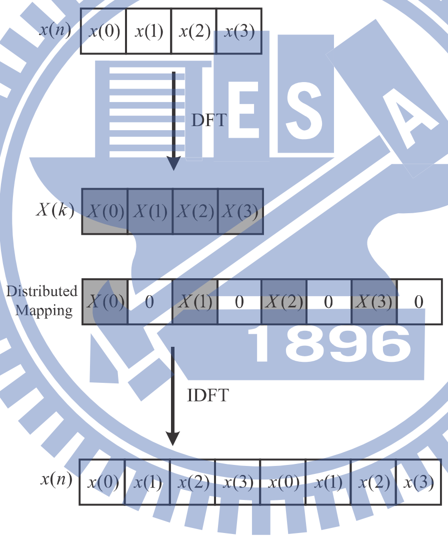
Figure 3.12: Distrubuted subcarrier mapper with N=6 mapping to M=8

Please note that there are some easier ways to generate the SC-FDMA signals for distributed subcarrier mapping. For instance, Fig. 3.13 shows that for N=M=8, there is no need to perform FFT and IFFT at all. Fig. 3.14 shows that for N=8 and M=4, the output signal of SC-FDMA can be obtained by repeating the input data twice so that there is no need to perform FFT and IFFT. In fact, there is no need to perform FFT and IFFT for distributed subcarrier mapping. Thus the proposed processor will be active only for the localized subcarrier mapping.



$x(n)$
$x(0)$ $x(1)$ $x(2)$ $x(3)$ $x(0)$ $x(1)$ $x(2)$ $x(3)$
DFT
$X(k)$
IDFT
$x(n)$
$x(0)$ $x(1)$ $x(2)$ $x(3)$ $x(0)$ $x(1)$ $x(2)$ $x(3)$

Figure 3.13: 8-point DFT and IDFT



$x(n)$
$x(0)$ $x(1)$ $x(2)$ $x(3)$
DFT
$X(k)$
$X(0)$ $X(1)$ $X(2)$ $X(3)$
Distributed Mapping
$X(0)$ 0 $X(1)$ 0 $X(2)$ 0 $X(3)$ 0
IDFT
$x(n)$
$x(0)$ $x(1)$ $x(2)$ $x(3)$ $x(0)$ $x(1)$ $x(2)$ $x(3)$

Figure 3.14: 4-point is mapping to 8-point by distrubuted subcarrier mapping

3.2.3 Ripple-like ON/OFF stage control

For FFT/IFFT hardware sharing, we should use ripple-like ON/OFF stage control. This can set each stage of the FFT processor to switch FFT or IFFT. In SC-FDMA uplink transmission, N-point FFT is mapped to M-point IFFT, where we always set M be 2048 and N be 128, 256, 512, 1024, 1536 and 2048. To reduce the hardware, we would like a FFT processing be capable to handle both FFT and IFFT processing. Thus, the mapped reordered FFT output data feed back to the input of the FFT processor. In this case, this FFT can control the reset time. It is shown in Fig. 3.15.

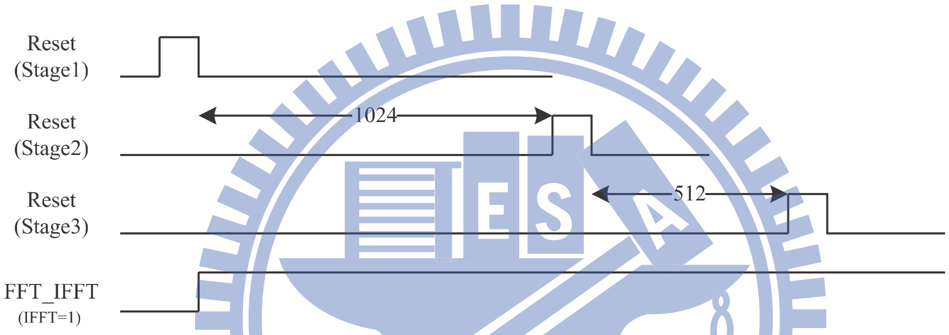
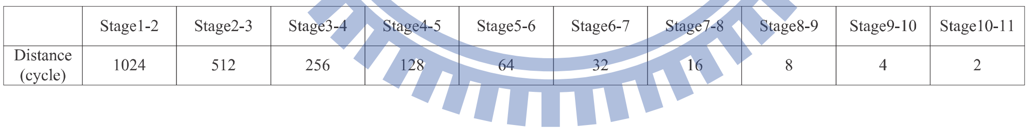


Figure 3.15: Clock cycle of ripple-like ON/OFF stage control



	Stage1-2	Stage2-3	Stage3-4	Stage4-5	Stage5-6	Stage6-7	Stage7-8	Stage8-9	Stage9-10	Stage10-11
Distance (cycle)	1024	512	256	128	64	32	16	8	4	2

Table 3.1: Distance for each stage

3.2.4 Early access of reordering

The mapped and reordered data can actually be accessed earlier and we do not need to wait until the whole mapping and reordering processes finish, Take a 8-point FFT output for instance, the output data are $X[0]$, $X[4]$, $X[2]$, $X[6]$, $X[1]$, $X[5]$, $X[3]$, $X[7]$. Thus we can read the output in when $X[1]$ is available, i.e. around half of the FFT size. For other FFT sizes, we summarize when the data can be accessed earlier in Table 3.2.

N-point	Time for early access
128	$7N/8$
256	$15N/16$
512	$15N/16$
1024	$31N/32$
1536	$31N/32$
2048	$31N/32$

Table 3.2: A fraction of N-point cab be read early

3.2.5 FFT/IFFT hardware sharing

In SC-FDMA system, we need to perform both FFT and IFFT. FFT and IFFT have similar structure except conjugate of input and output data. The Fig. 3.16 shows a timing plan for a N-point SC-FDMA system, we expect each FFT/IFFT or reordering takes N clock cycles. FFT/IFFT latency is N and reordering latency is N, thus we need $4N$ clock cycles. As a result, if the required data is R, we should use $4R$ for the proposed FFT/IFFT precessor.

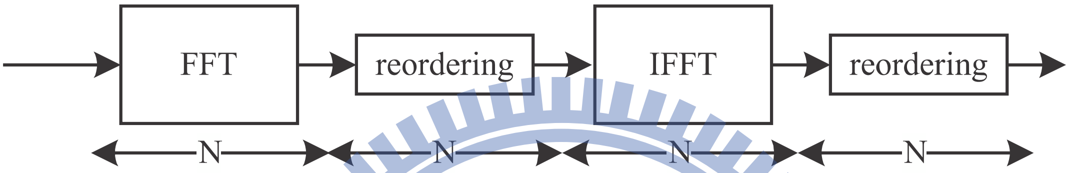
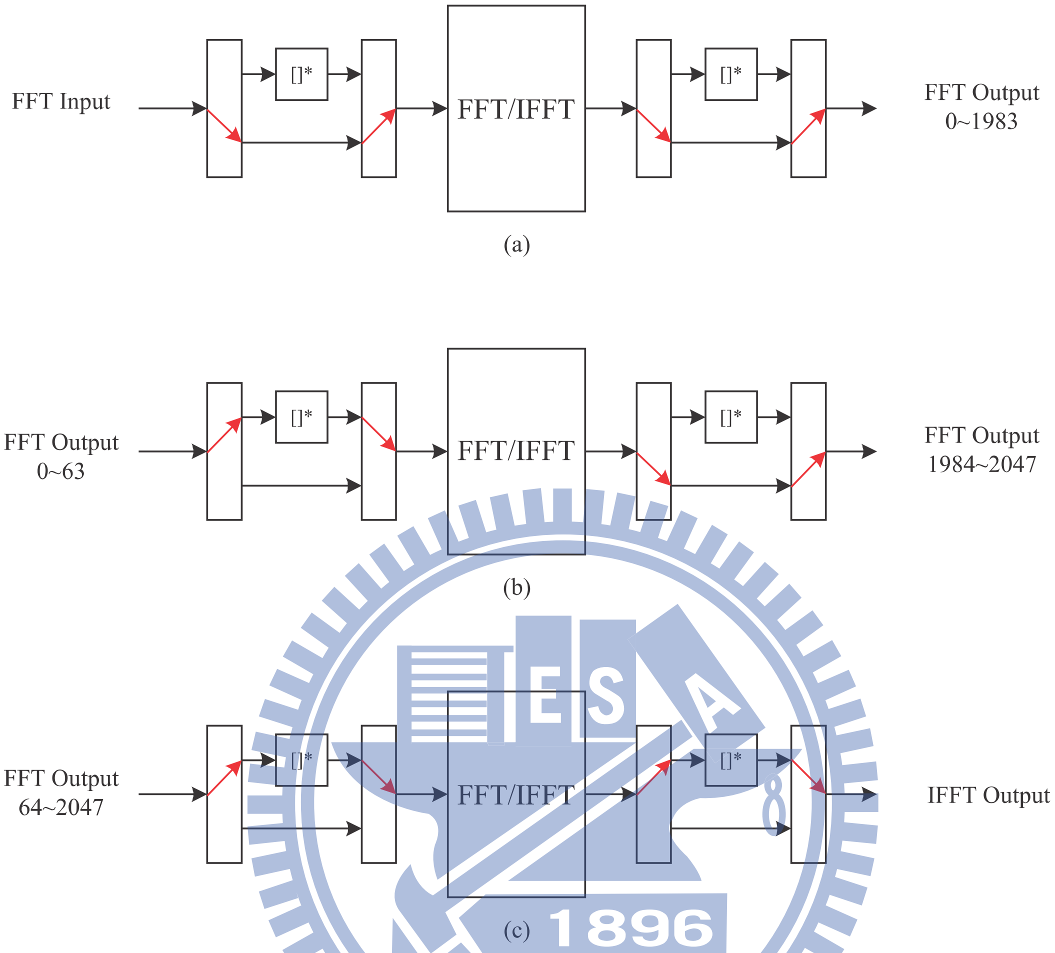


Figure 3.16: Block diagram of FFT/IFFT for SC-FDMA system

The proposed processor uses pipeline, ripple-like ON/OFF stage control and early access of reordering to reduce the hardware. An operation example for N=2048 is shown in Figs. 3.17 and 3.18. Please note that the (a), (b) and (c) correspond to Figs. 3.17 (a), (b) and (c). When the clock cycles are $2048 + 1984(2048 * 31/32) = 4033$, which is shown in Fig. 3.18, IFFT starts to operate in Fig. 3.17(b). Fig. 3.17(c) is the IFFT operation.



(a)

(b)

(c)

Figure 3.17: (a) FFT operation (b) Switch FFT to IFFT operation (early access of reordering and ripple-like ON/OFF stage control start) (c) IFFT operation

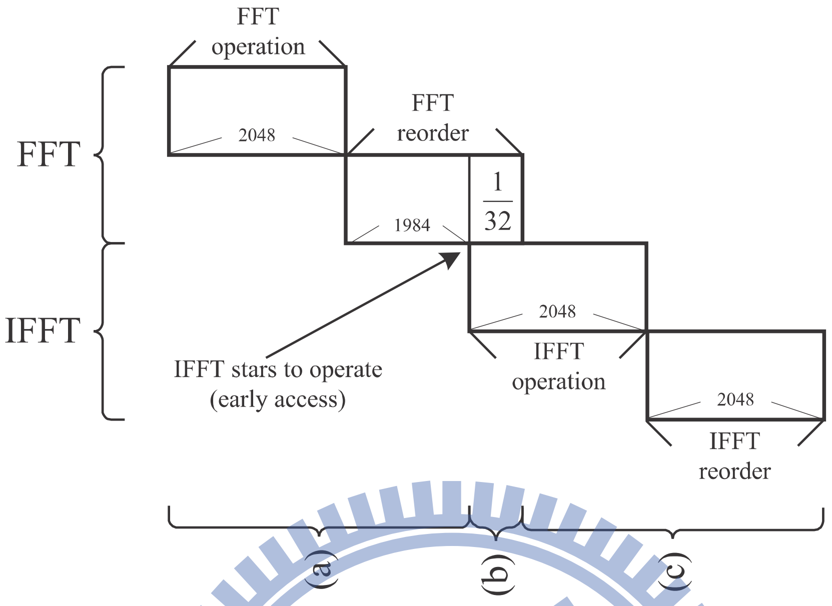


Figure 3.18: Timing planning of FFT/IFFT for $N = 2048$

3.3 Complexity analysis

Because SC-FDMA system has both FFT and IFFT, but we combine them in one. We almost reduce a half of the hardware, the requirement hardware is shown in Table 3.3.

	Complex multiplier	Complex adder	ROM table(bytes)	Memory size(bytes)
Proposes FFT/IFFT	7	25	6144	6142

Table 3.3: Comparison of hardware requirement.

3.4 Simulation

We conduct two simulation in MATLAB before we design our proposed FFT/IFFT, i.e. SQNR (Signal to Quantization Noise Ratio) simulation and FFT/IFFT loop function test. SQNR simulation can determine the bit width of the FFT, and FFT/IFFT loop function test can verify the function correctness of the SC-FDMA system.

3.4.1 SQNR (Signal to Quantization Noise Ratio) simulation

Determining appropriate bit width in the FFT processor is important. Since the bit width affects the hardware cost directly. Bit width can be determine by fixed-point simulation. We use the SQNR (Signal to Quantization Noise Ratio) system model to determine the FFT bit width. The system model of SQNR is shown below.

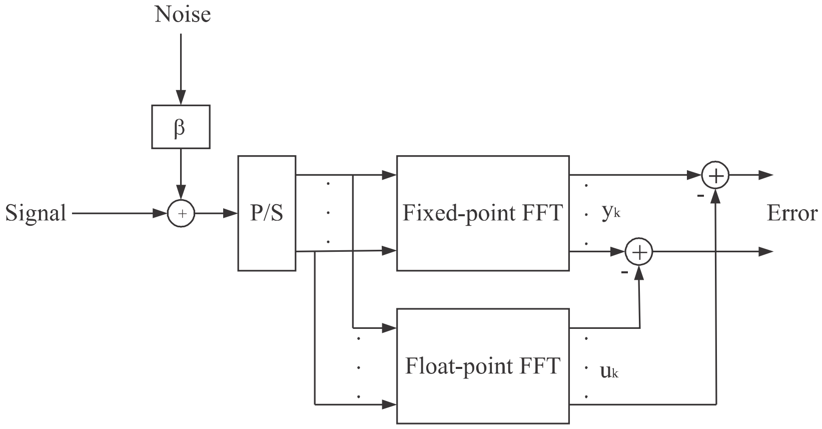


Figure 3.19: System model of SQNR.

The SQNR is defined as

$$SQNR = \frac{1}{N}\sum_{k=0}^{N-1}\frac{\sigma_x^2}{|y_k - u_k| + \beta^2\sigma_e^2} \tag{3.12}$$

Consider the SNR (Signal to Noise Ratio) is one ($\frac{\sigma_x^2}{\sigma_e^2} = 1$), we can rewrite the equation as

$$SQNR = \frac{1}{N}\sum_{k=0}^{N-1}\frac{1}{|y_k - u_k| + \beta^2} , \tag{3.13}$$

where $\beta = 10^{\frac{-SNR}{20}}$. The relationship between SQNR (with final bit-width) and SNR is shown in Fig. 3.20.

When SQNR is large, it means that quantization error is small. From the figure, the final bit-width can support SQNR up to 40 dB.

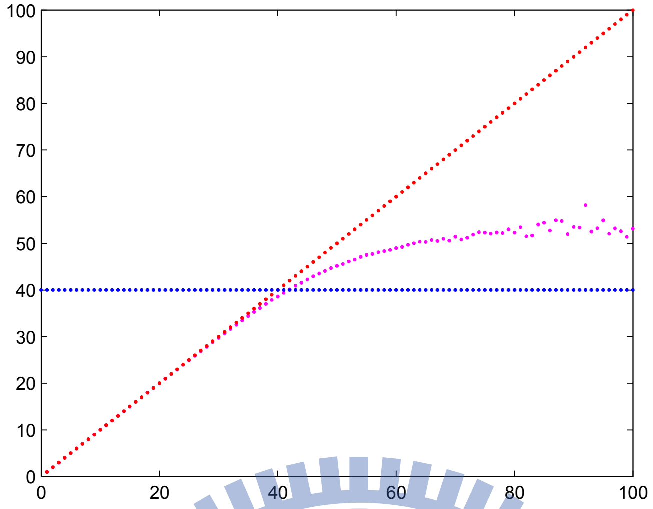


Figure 3.20: SQNR v.s. SNR.

3.4.2 FFT/IFFT loop function test

For verifing SC-FDMA system, we use MATLAB to simulate a loop which cascading FFT and IFFT as shown in Fig. 3.21. Fig. 3.22 is shown the comparison of fixed-point system platform and floating-point system platform. Fig. 3.23 shows the comparison of input and fixed-point system output4 and floating-point system output4. Fig. 3.24 shows the comparison of input and fixed-point system output2 and output4.

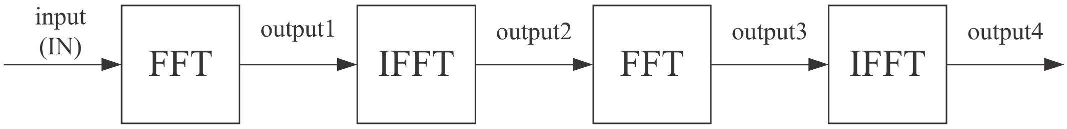


Figure 3.21: Block diagram of FFT and IFFT in SC-FDMA system

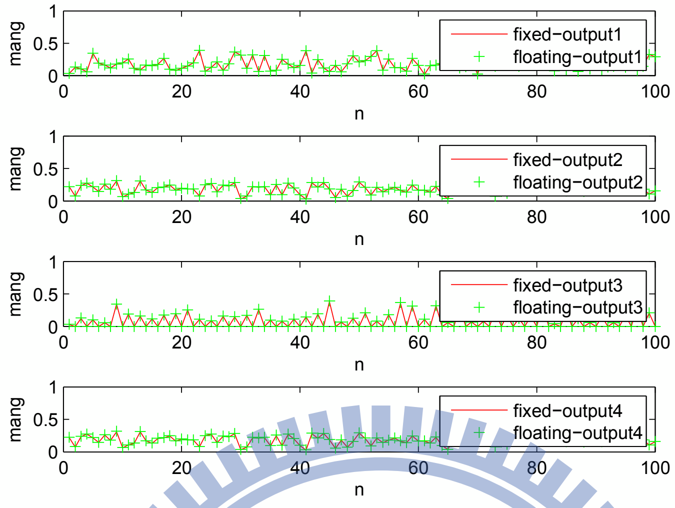


Figure 3.22: fixed-point v.s. floating-point

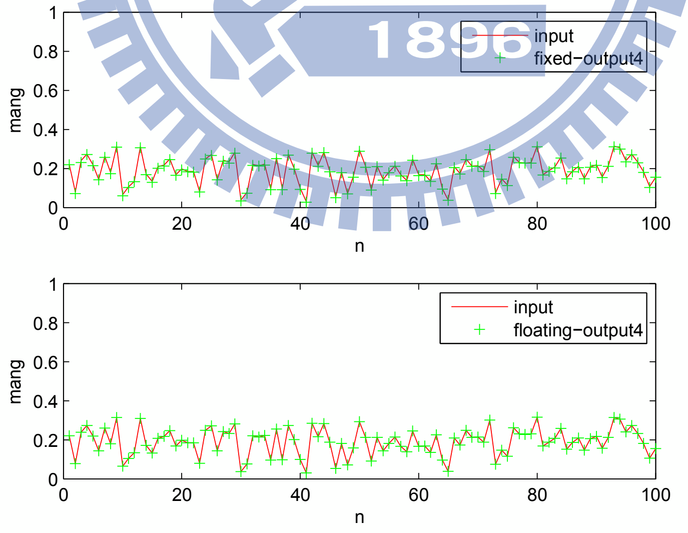


Figure 3.23: Input v.s. fixed-point output4 and floating-point output4

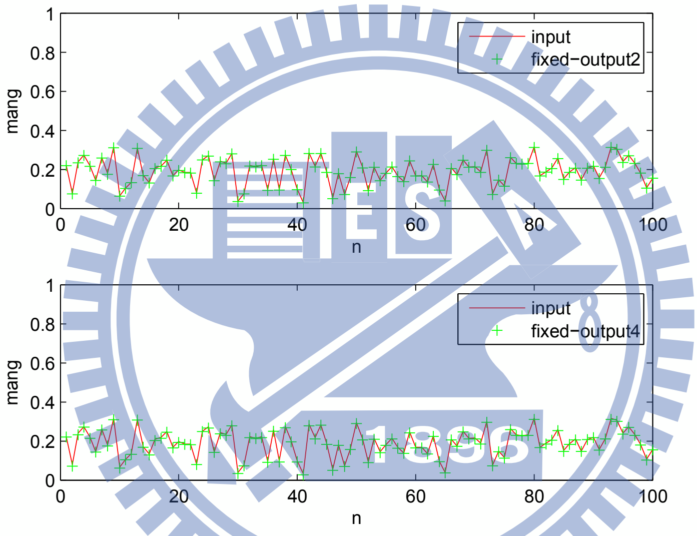


Figure 3.24: Input v.s. fixed-point output2 and fixed-point output4

Chapter 4

FPGA implementation and verification

In this chapter we state how to build the MATLAB platform of the variable FFT and how to verify the design. The MATLAB platform can be used to verify the function of the RTL platform. Then we use FPGA to verify the output data of the RTL platform. The design flow is illustrate in Fig. 4.1.

- **RTL verification** We use MATLAB to analyse and verify architecture and use verilog to design architecture.
- **Synthesis and Place and Route** We add ICON and ILA of Chipscope and .ucf file which is pin location file in Xilinx synthesis.
- **Xilinx iMPACT** We can use iMPACT to download .bit file or .mcs file in FPGA to verify our design.

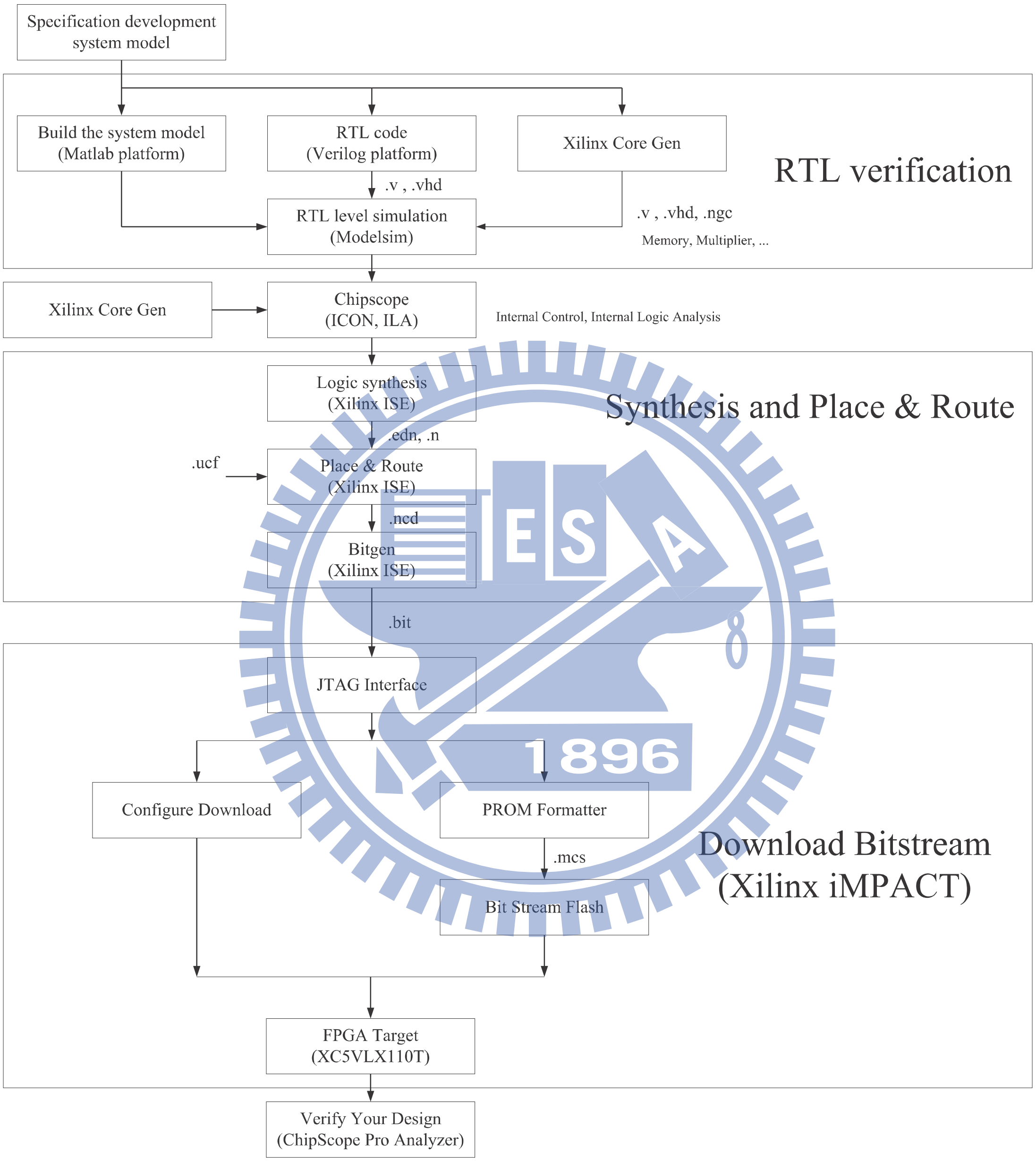


Figure 4.1: A design flow of the Xilinx design kit Vertex5-XC5VLX110T-FF1136

4.1 RTL verification

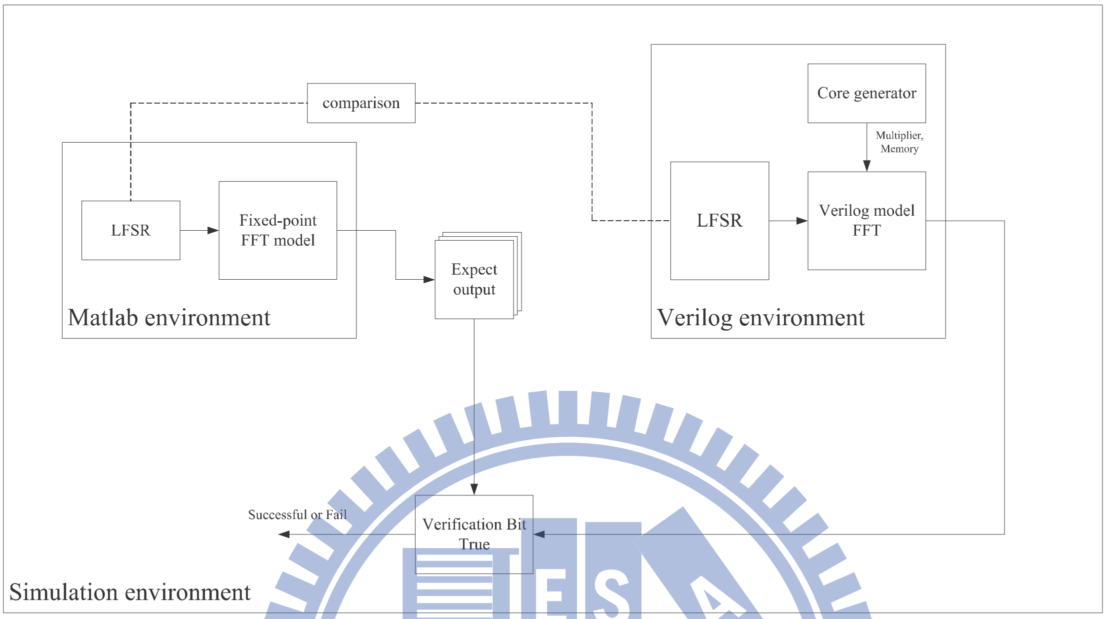
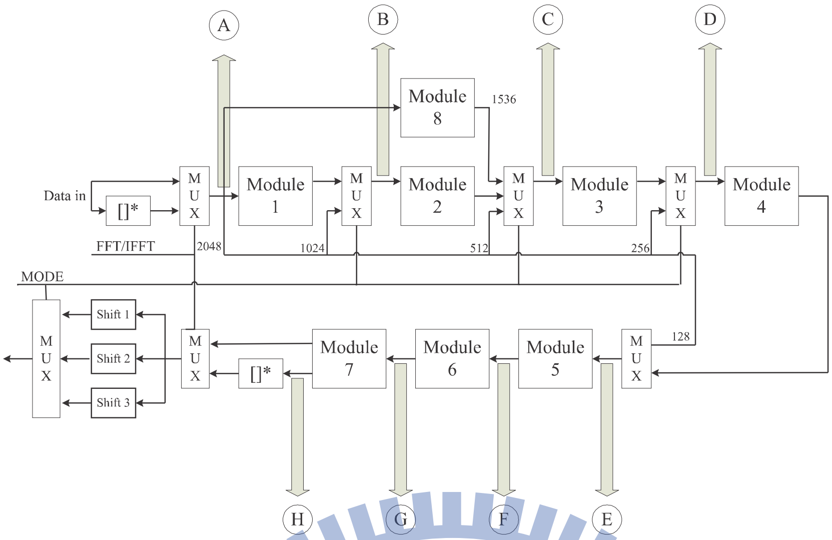


Figure 4.2: Simulation environment for variable FFT

We need to build the MATLAB platform and verilog platform. Fig. 4.2 shows the simulation environment for bit-true verification. At first we need to develop a floating-point FFT model in MATLAB environment. Since the FFT function in MATLAB tool is a floating-point function, we need to build the MATLAB quantizable FFT function using the proposed architecture mentioned earlier. When the quantizable FFT function is ready, we can determine the bit-width for individual stages using the procedure mentioned in Sec. 3.2.6. Fig. 4.3 shows the quantized result in all stages. Note that total number of bits in all stage is 16. When the fixed-point FFT model is ready, we use LFSR (Liner feedback shift register) to generate random input data pattern from this model.

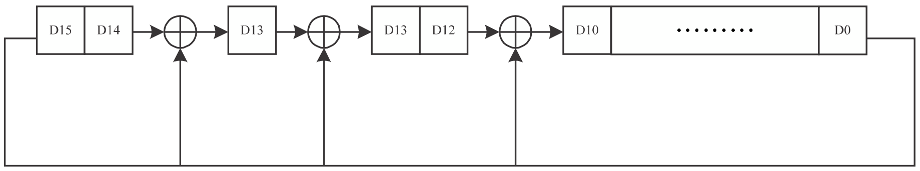


Point	A	B	C	D	E	F	G	H
Integer bit	2	3	3	4	4	5	6	7

Figure 4.3: Bit-width in all stages

4.1.1 LFSR (Liner feedback shift register)

Usually, the random input data patten of MATLAB will be used by Verilog, but we will verify that in FPGA. Thus we use LFSR to implement the random input data both in MATLAB and Verilog. First we confirm the two input data of MATLAB platform and Verilog platform are the same, and then we compare the output data from the verilog platform and the MATLAB platform to verify the result. The LFSR will be inserted in FPGA with FFT. Thus, we can verify the output data in FPGA. Fig. 4.6 shows the used LFSR structure with 16 bits.



$$EQ = X^{16} + X^{14} + X^{13} + X^{11} + 1$$

Figure 4.4: LFSR for 16 bits

Since the data rate is R and the operation rate is $4R$, we need four LFSR in simulation. which is shown inFig. 4.5. They generate four input data simultaneously. So there are four input data waiting for the FFT processor. By doing this, the timing diagram of the input data is shown in Fig. 4.6.

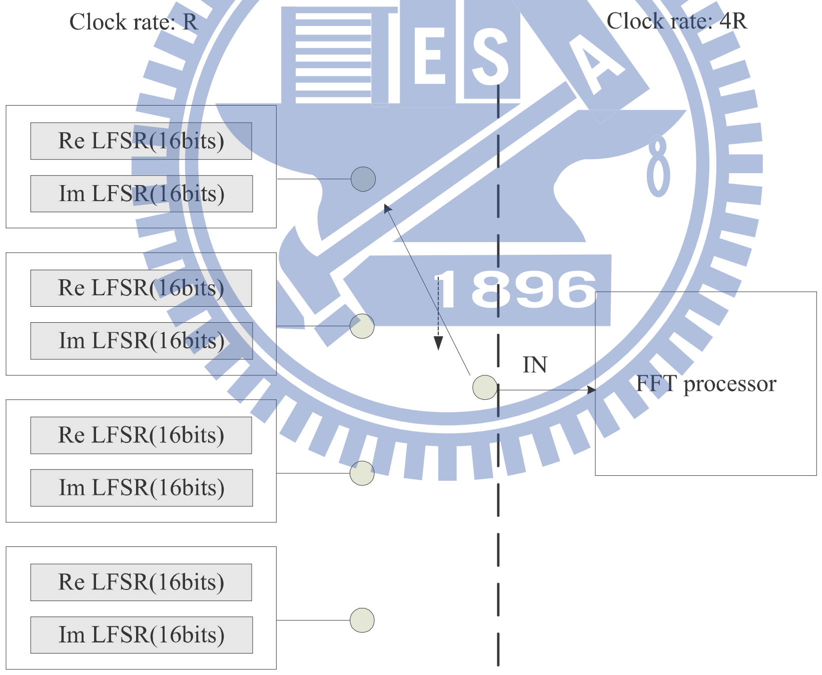


Figure 4.5: LFSR and FFT processor

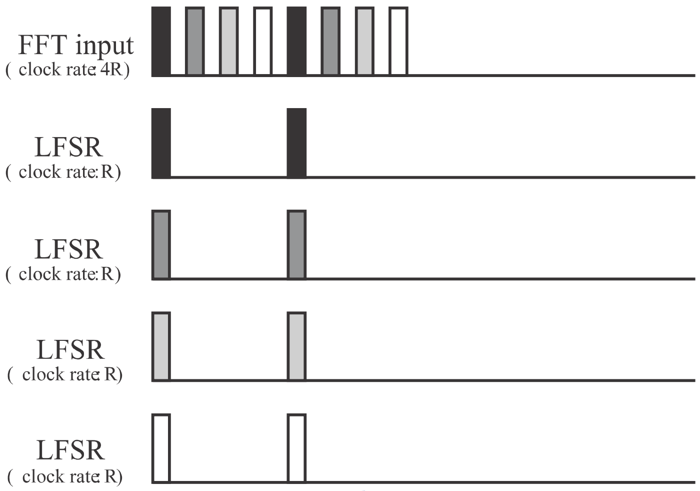


Figure 4.6: Timing plan for LFSR

4.2 Synthesis place and route

- **Overview of ISE software** [9]

  When RTL code is ready, we will use ISE software to synthesis. The ISE software controls all aspects of the design flow. Through the Project Navigator interface, we can access all of the design entry and design implementation tools. You can also access the files and documents associated with the project. We will conduct the operation steps as follows.

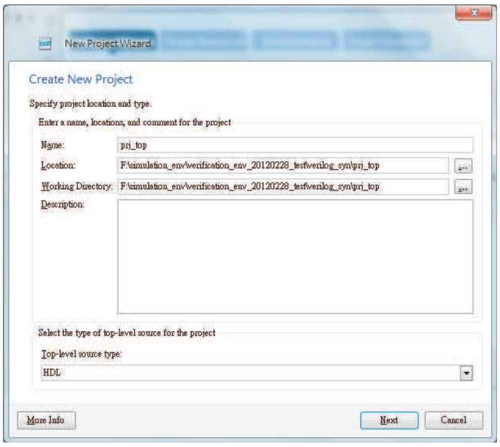
- **Starting the ISE software**

  To start the ISE software, double-click the ISE Project Navigator icon on your desktop.

- **Creating a new project**

  To create a new project using the New Project Wizard, do the following

From Project Navigator, select File → New Project. The New Project Wizard appears.



Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: prj_top

Location: F:\simulation_env\verification_env_20120228_test\verilog_syn\prj_top

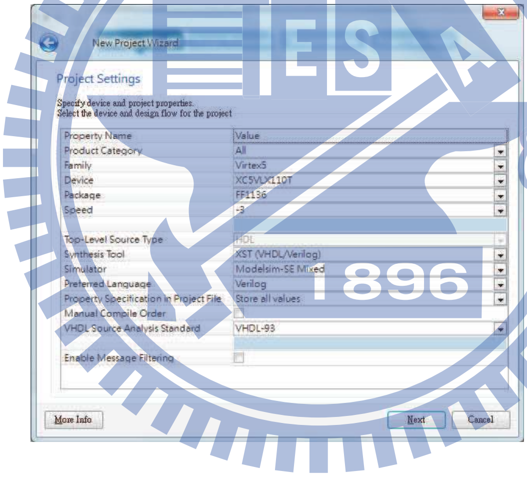
Working Directory: F:\simulation_env\verification_env_20120228_test\verilog_syn\prj_top

Description:

Select the type of top-level source for the project

Top-level source type: HDL

In the Location field, browse to the directory in which we installed the project, and write the file name. Click Next and the New Project Wizard page appears.



Project Settings

Specify device and project properties.
Select the device and design flow for the project

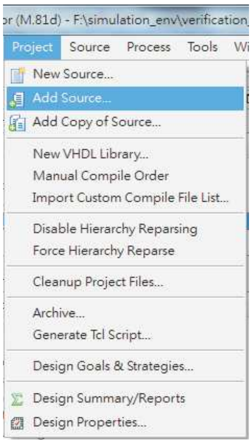
Property Name	Value
Product Category	All
Family	Virtex5
Device	XC5VLX110T
Package	FF1136
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE Mixed
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	□
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	□

Select the FPGA types in the New Project WizardXDevice Properties page , and click Next. Our FPGA is Vertex5-XC5VLX110T-FF1136

- **Adding source files**

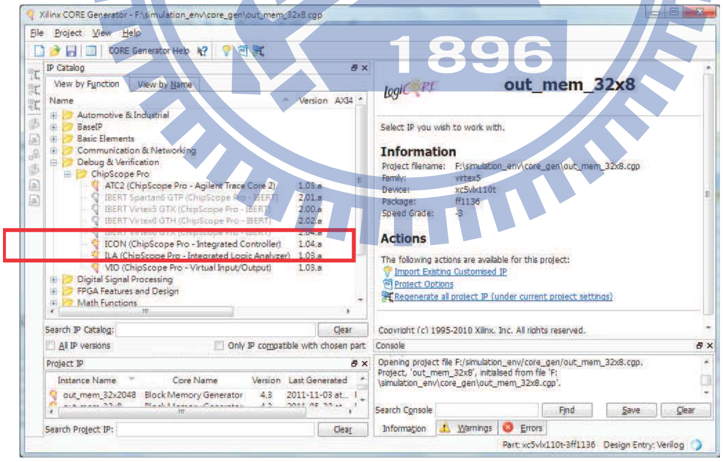
  HDL files have to be added to the project before they can be synthesized.

Select Project → Add Source, and Select the following files (.vhd files for VHDL design entry or .v files for Verilog design entry) from the project directory, and click Open.



- **Overview of CORE Generator software** [9]
  The CORE Generator software is a graphical interactive design tool that enables we to create high-level modules such as memory elements, math functions and communications, and I/O interface cores.

We can use CORE generator to generate ICON and ILA of Chipscope, and then add source of them.


Xilinx CORE Generator - F:\simulation_env\core_gen\out_mem_32x8.cgp
File
Project
View
Help
CORE Generator Help
IP Catalog
View by Function
View by Name
Name
Version
AXI4
Automotive & Industrial
BaseIP
Basic Elements
Communication & Networking
Debug & Verification
ChipScope Pro
ATC2 (ChipScope Pro - Agilent Trace Core 2) 1.03.a
IBERT Spartan6 GTP (ChipScope Pro - IBERT) 2.01.a
IBERT Virtex5 GTX (ChipScope Pro - IBERT) 2.00.a
IBERT Virtex6 GTH (ChipScope Pro - IBERT) 2.02.a
ICON (ChipScope Pro - Integrated Controller) 1.04.a
ILA (ChipScope Pro - Integrated Logic Analyzer) 1.03.a
VIO (ChipScope Pro - Virtual Input/Output) 1.03.a
Digital Signal Processing
FPGA Features and Design
Math Functions
Search IP Catalog:
Clear
All IP versions
Only IP compatible with chosen part
Project IP
Search Project IP:
Clear
LogiCORE
out_mem_32x8
Actions
Import Existing Customised IP
Project Options
Regenerate all project IP (under current project settings)
Console
Search Console
Find
Save
Clear
Information
Warnings
Errors

Select IP you wish to work with.

Information

Project filename: F:\simulation_env\core_gen\out_mem_32x8.cgp
Family: virtex5
Device: xc5vlx110t
Package: ff1136
Speed Grade: -3

The following actions are available for this project:

Instance Name	Core Name	Version	Last Generated
out_mem_32x2048	Block Memory Generator	4.3	2011-11-03 at...

Copyright (c) 1995-2010 Xilinx, Inc. All rights reserved.

```
Opening project file F:/simulation_env/core_gen/out_mem_32x8.cgp.
Project, 'out_mem_32x8', initialised from file 'F:
\simulation_env\core_gen\out_mem_32x8.cgp'.
```

Part: xc5vlx110t-3ff1136 Design Entry: Verilog

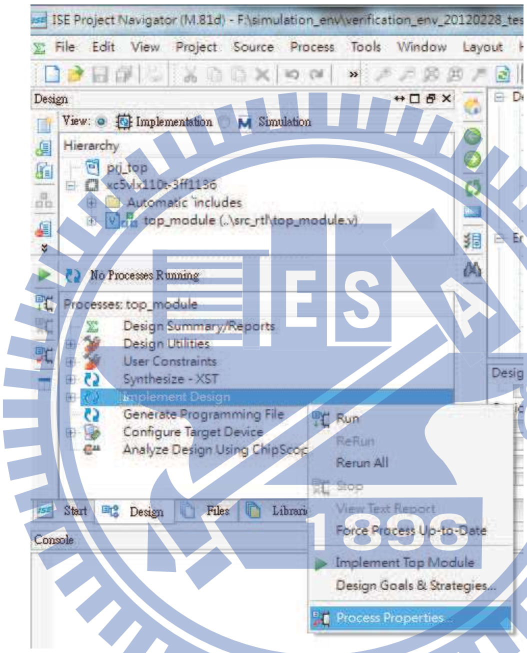
- **Chipscope**
  LA (logic analzer) is an important tool of FPGA, but traditional LA has two disadvantages, which are expensive and need more pads. Thus the

Xilinx provide a tool called "ChipScope Pro". ChipScope Pro is cheap and can be opearated easily.

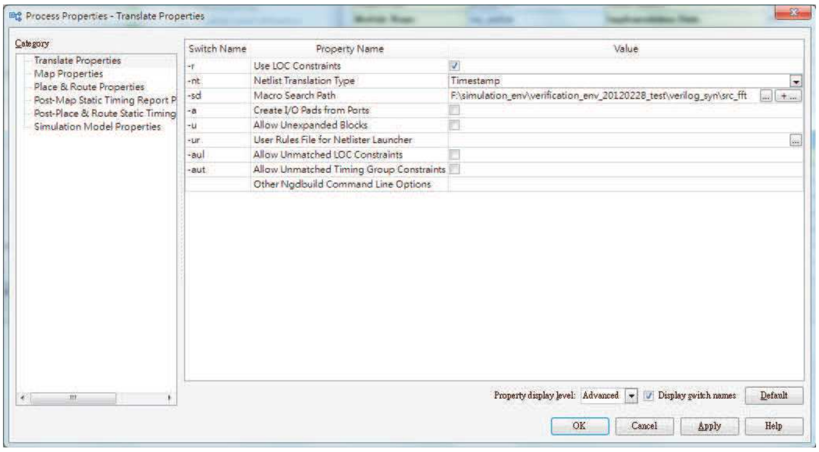
- **Setting macro search path**

  If we use CORE generator to generate memory, mutiplier or else module, we have to set macro search path.

In the Processes Pane → Implement Design, right Click → Process Properties.



Set macro search path that choose the path in which our CORE generator file and then click OK.



Process Properties - Translate Properties

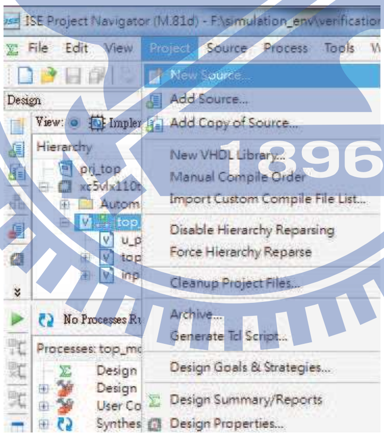
Category
- Translate Properties
- Map Properties
- Place & Route Properties
- Post-Map Static Timing Report P
- Post-Place & Route Static Timing
- Simulation Model Properties

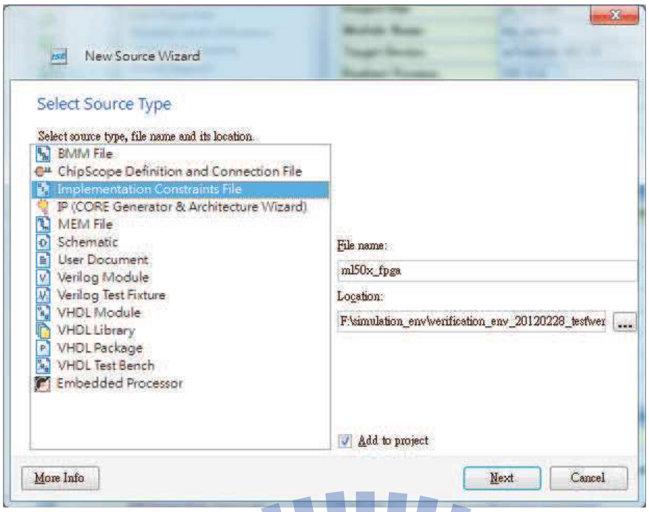
Switch Name	Property Name	Value
-r	Use LOC Constraints	☑
-nt	Netlist Translation Type	Timestamp
-sd	Macro Search Path	F:\simulation_env\verification_env_20120228_test\verilog_syn\src_fft
-a	Create I/O Pads from Ports	□
-u	Allow Unexpanded Blocks	□
-ur	User Rules File for Netlister Launcher	
-aul	Allow Unmatched LOC Constraints	□
-aut	Allow Unmatched Timing Group Constraints	□
	Other Ngdbuild Command Line Options	

Property display level: Advanced ☑ Display switch names

- **Create constraint file (.ucf file)**

Select Project → New Source → Implementation Constrains File, and then set file name and location.





New Source Wizard

Select Source Type

Select source type, file name and its location.

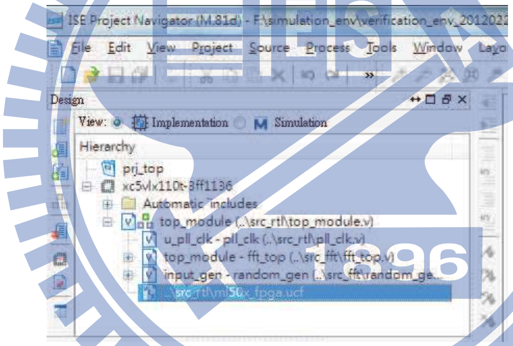
- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: ml50x_fpga

Location: F:\simulation_env\verification_env_20120228_testver

☑ Add to project

The constrain file is in current hierarchy.



ISE Project Navigator (M.81d) - F:\simulation_env\verification_env_2012022

View: ● Implementation ○ Simulation

```
Hierarchy
  prj_top
⊟ xc5vlx110t-3ff1136
    ⊞ Automatic `includes
    ⊟ top_module (..\src_rtl\top_module.v)
          u_pll_clk - pll_clk (..\src_rtl\pll_clk.v)
        ⊞ top_module - fft_top (..\src_fft\fft_top.v)
        ⊞ input_gen - random_gen (..\src_fft\random_ge...
          ..\src_rtl\ml50x_fpga.ucf
```

- **Edit constraint file**

Setting clock period and location. (Refer to Xilinx tutorial ug347)

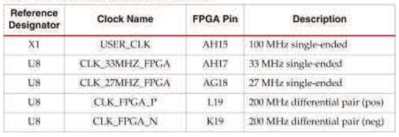
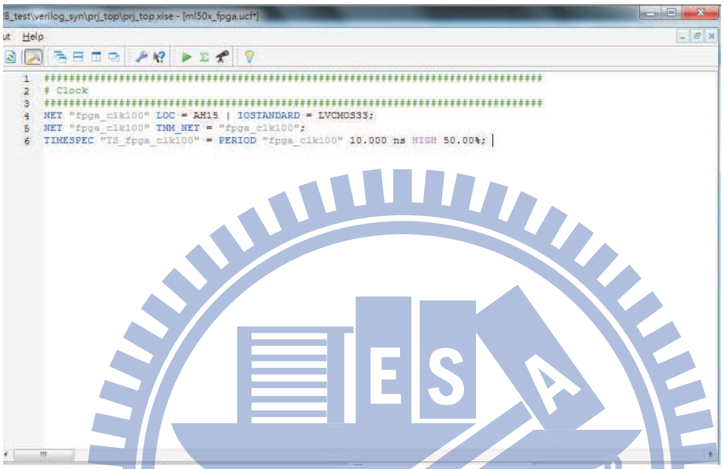


Table 1-4: **Oscillator Socket Connections**

Reference Designator	Clock Name	FPGA Pin	Description
X1	USER_CLK	AH15	100 MHz single-ended
U8	CLK_33MHZ_FPGA	AH17	33 MHz single-ended
U8	CLK_27MHZ_FPGA	AG18	27 MHz single-ended
U8	CLK_FPGA_P	L19	200 MHz differential pair (pos)
U8	CLK_FPGA_N	K19	200 MHz differential pair (neg)



```
##############################################################################
# Clock
##############################################################################
NET "fpga_clk100" LOC = AH15 | IOSTANDARD = LVCMOS33;
NET "fpga_clk100" TNM_NET = "fpga_clk100";
TIMESPEC "TS_fpga_clk100" = PERIOD "fpga_clk100" 10.000 ns HIGH 50.00%;
```

Setting DIP switch location. (Refer to Xilinx tutorial ug347)

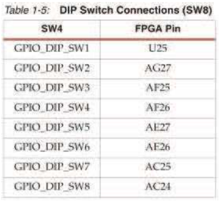
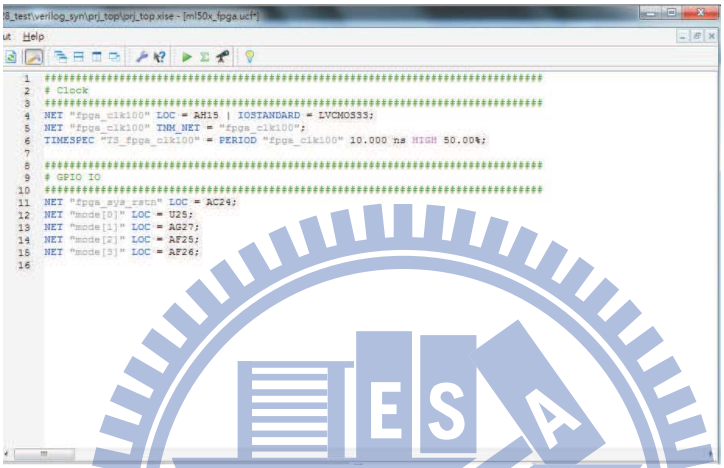


Table 1-5: **DIP Switch Connections (SW8)**

SW4	FPGA Pin
GPIO_DIP_SW1	U25
GPIO_DIP_SW2	AG27
GPIO_DIP_SW3	AF25
GPIO_DIP_SW4	AF26
GPIO_DIP_SW5	AE27
GPIO_DIP_SW6	AE26
GPIO_DIP_SW7	AC25
GPIO_DIP_SW8	AC24



```
##############################################################################
# Clock
##############################################################################
NET "fpga_clk100" LOC = AH15 | IOSTANDARD = LVCMOS33;
NET "fpga_clk100" TNM_NET = "fpga_clk100";
TIMESPEC "TS_fpga_clk100" = PERIOD "fpga_clk100" 10.000 ns HIGH 50.00%;

##############################################################################
# GPIO IO
##############################################################################
NET "fpga_sys_rstn" LOC = AC24;
NET "mode[0]" LOC = U25;
NET "mode[1]" LOC = AG27;
NET "mode[2]" LOC = AF25;
NET "mode[3]" LOC = AF26;

```

Setting LED location. (Refer to Xilinx tutorial ug347)

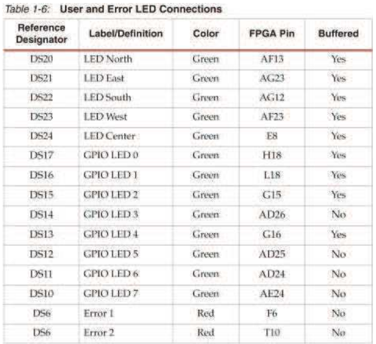
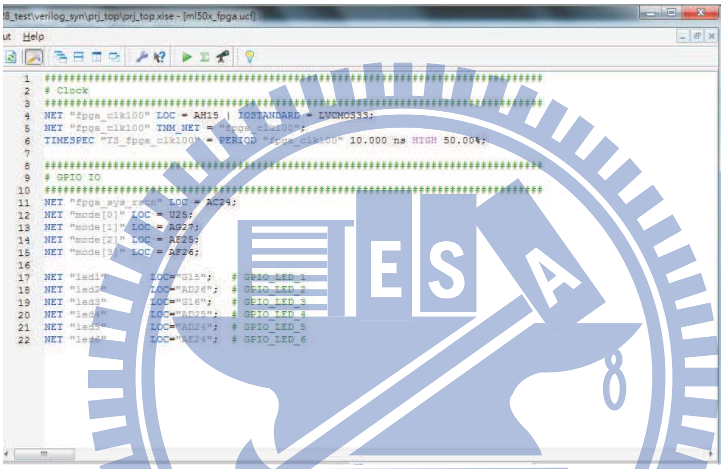


Table 1-6: **User and Error LED Connections**

Reference Designator	Label/Definition	Color	FPGA Pin	Buffered
DS20	LED North	Green	AF13	Yes
DS21	LED East	Green	AG23	Yes
DS22	LED South	Green	AG12	Yes
DS23	LED West	Green	AF23	Yes
DS24	LED Center	Green	E8	Yes
DS17	GPIO LED 0	Green	H18	Yes
DS16	GPIO LED 1	Green	L18	Yes
DS15	GPIO LED 2	Green	G15	Yes
DS14	GPIO LED 3	Green	AD26	No
DS13	GPIO LED 4	Green	G16	Yes
DS12	GPIO LED 5	Green	AD25	No
DS11	GPIO LED 6	Green	AD24	No
DS10	GPIO LED 7	Green	AE24	No
DS6	Error 1	Red	F6	No
DS6	Error 2	Red	T10	No



```
##############################################################################
# Clock
##############################################################################
NET "fpga_clk100" LOC = AH15 | IOSTANDARD = LVCMOS33;
NET "fpga_clk100" TNM_NET = "fpga_clk100";
TIMESPEC "TS_fpga_clk100" = PERIOD "fpga_clk100" 10.000 ns HIGH 50.00%;

##############################################################################
# GPIO IO
##############################################################################
NET "fpga_sys_rstn" LOC = AC24;
NET "mode[0]" LOC = U25;
NET "mode[1]" LOC = AG27;
NET "mode[2]" LOC = AF25;
NET "mode[3]" LOC = AF26;

NET "led1"      LOC="G15";   # GPIO_LED_1
NET "led2"      LOC="AD26";  # GPIO_LED_2
NET "led3"      LOC="G16";   # GPIO_LED_3
NET "led4"      LOC="AD25";  # GPIO_LED_4
NET "led5"      LOC="AD24";  # GPIO_LED_5
NET "led6"      LOC="AE24";  # GPIO_LED_6
```

Setting press location for reprogramming FPGA. (Refer to Xilinx tutorial ug347)

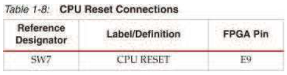
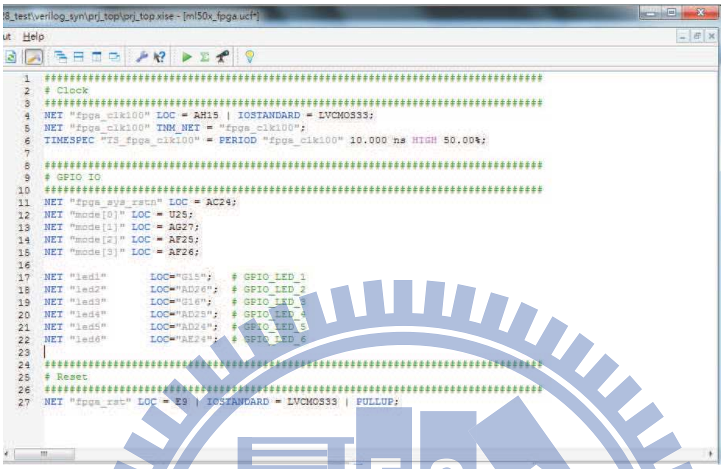


Table 1-8: **CPU Reset Connections**

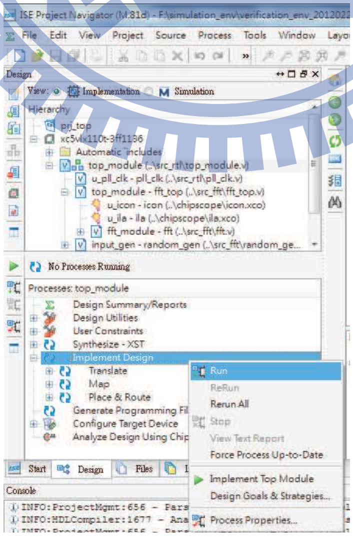
Reference Designator	Label/Definition	FPGA Pin
SW7	CPU RESET	E9



```
################################################################################
# Clock
################################################################################
NET "fpga_clk100" LOC = AH15 | IOSTANDARD = LVCMOS33;
NET "fpga_clk100" TNM_NET = "fpga_clk100";
TIMESPEC "TS_fpga_clk100" = PERIOD "fpga_clk100" 10.000 ns HIGH 50.00%;

################################################################################
# GPIO IO
################################################################################
NET "fpga_sys_rstn" LOC = AC24;
NET "mode[0]" LOC = U25;
NET "mode[1]" LOC = AG27;
NET "mode[2]" LOC = AF25;
NET "mode[3]" LOC = AF26;

NET "led1"        LOC="G15";   # GPIO_LED_1
NET "led2"        LOC="AD26";  # GPIO_LED_2
NET "led3"        LOC="G16";   # GPIO_LED_3
NET "led4"        LOC="AD25";  # GPIO_LED_4
NET "led5"        LOC="AD24";  # GPIO_LED_5
NET "led6"        LOC="AE24";  # GPIO_LED_6

################################################################################
# Reset
################################################################################
NET "fpga_rst" LOC = E9 | IOSTANDARD = LVCMOS33 | PULLUP;
```

- **Synthesis and implement design**

In the Processes Pane → Implement Design, right-click → Run



ISE Project Navigator (M.81d) - F:\simulation_env\verification_env_2012022
File Edit View Project Source Process Tools Window Layo
Design
View: Implementation Simulation
Hierarchy
prj_top
xc5vlx110t-3ff1136
Automatic `includes
top_module (..\src_rtl\top_module.v)
u_pll_clk - pll_clk (..\src_rtl\pll_clk.v)
top_module - fft_top (..\src_fft\fft_top.v)
u_icon - icon (..\chipscope\icon.xco)
u_ila - ila (..\chipscope\ila.xco)
fft_module - fft (..\src_fft\fft.v)
input_gen - random_gen (..\src_fft\random_ge...
No Processes Running
Processes: top_module
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Programming Fil
Configure Target Device
Analyze Design Using Chip
Run
ReRun
Rerun All
Stop
View Text Report
Force Process Up-to-Date
Implement Top Module
Design Goals & Strategies...
Process Properties...
Start
Design
Files
Console

```
INFO:ProjectMgmt:656 - Pars
INFO:HDLCompiler:1677 - Ana
```

4.3 Xilinx iMPACT

[9]

IMPACT is a file generation and device programming tool. iMPACT enables you to program through several parallel cables, including the Platform Cable USB. iMPACT can create bitstream files, System ACE solution files, PROM files, and SVF/XSVF files.

- **Platform cable USB-II**

  The Platform Cable connects to the USB port of your computer and can be used to facilitate Boundary-Scan functionality. For more information, see the Platform Cable USB-II Data Sheet available from the Xilinx website.

- **Generating the configuration files**

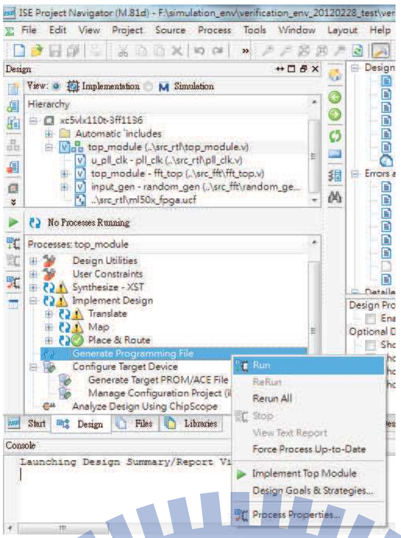
  BIT file: A binary file that contains proprietary header information as well as configuration data.(We choose BIT file to do our verifycation.)

  MCS file: An ASCII file that contains PROM configuration information.

We choose BIT file in our simulation.

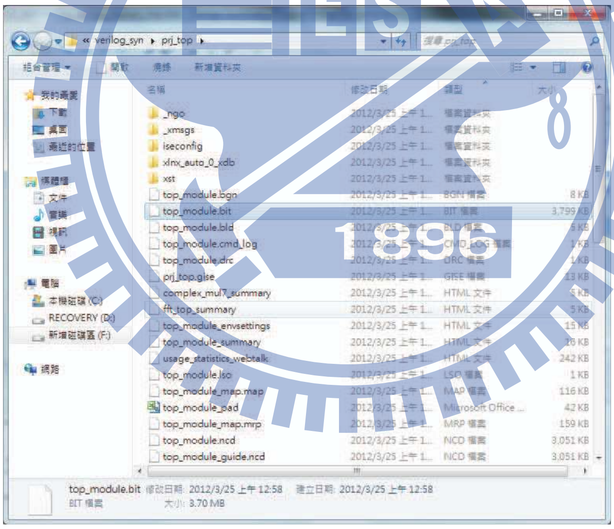
- **Generate .bit file**

In the Processes Pane → Implement Design → Generate Programming File, right-click → Run


ISE Project Navigator (M.81d) - F:\simulation_env\verification_env_20120228_test\ver
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
xc5vlx110t-3ff1136
Automatic `includes
top_module (..\src_rtl\top_module.v)
u_pll_clk - pll_clk (..\src_rtl\pll_clk.v)
top_module - fft_top (..\src_fft\fft_top.v)
input_gen - random_gen (..\src_fft\random_ge...
..\src_rtl\ml50x_fpga.ucf
No Processes Running
Processes: top_module
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Generate Target PROM/ACE File
Manage Configuration Project
Analyze Design Using ChipScope
Start
Design
Files
Libraries
Console
Run
ReRun
Rerun All
Stop
View Text Report
Force Process Up-to-Date
Implement Top Module
Design Goals & Strategies...
Process Properties...
Design
Errors
Detailed
Design Pro
Optional D

```
Launching Design Summary/Report Vi
```

We can find .bit file in our created project.

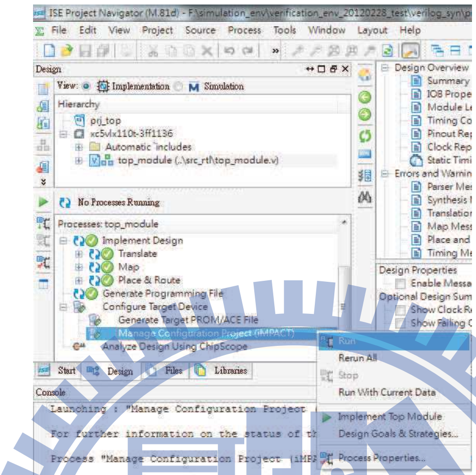


名稱	修改日期	類型	大小
_ngo	2012/3/25 上午 1...	檔案資料夾	
_xmsgs	2012/3/25 上午 1...	檔案資料夾	
iseconfig	2012/3/25 上午 1...	檔案資料夾	
xlnx_auto_0_xdb	2012/3/25 上午 1...	檔案資料夾	
xst	2012/3/25 上午 1...	檔案資料夾	
top_module.bgn	2012/3/25 上午 1...	BGN 檔案	8 KB
top_module.bit	2012/3/25 上午 1...	BIT 檔案	3,799 KB
top_module.bld	2012/3/25 上午 1...	BLD 檔案	5 KB
top_module.cmd_log	2012/3/25 上午 1...	CMD_LOG 檔案	1 KB
top_module.drc	2012/3/25 上午 1...	DRC 檔案	1 KB
prj_top.gise	2012/3/25 上午 1...	GISE 檔案	13 KB
complex_mul7_summary	2012/3/25 上午 1...	HTML 文件	5 KB
fft_top_summary	2012/3/25 上午 1...	HTML 文件	5 KB
top_module_envsettings	2012/3/25 上午 1...	HTML 文件	15 KB
top_module_summary	2012/3/25 上午 1...	HTML 文件	16 KB
usage_statistics_webtalk	2012/3/25 上午 1...	HTML 文件	242 KB
top_module.lso	2012/3/25 上午 1...	LSO 檔案	1 KB
top_module_map.map	2012/3/25 上午 1...	MAP 檔案	116 KB
top_module_pad	2012/3/25 上午 1...	Microsoft Office ...	42 KB
top_module_map.mrp	2012/3/25 上午 1...	MRP 檔案	159 KB
top_module.ncd	2012/3/25 上午 1...	NCD 檔案	3,051 KB
top_module_guide.ncd	2012/3/25 上午 1...	NCD 檔案	3,051 KB

top_module.bit 修改日期: 2012/3/25 上午 12:58 建立日期: 2012/3/25 上午 12:58
BIT 檔案 大小: 3.70 MB

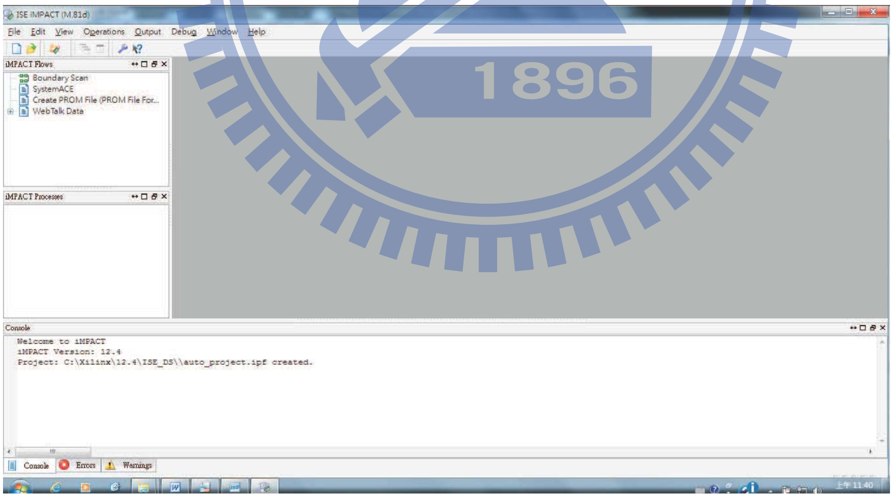
- **Download .bit file**

In the Processes Pane → Manage Configuration Project, right-click → Run (Open iMPACT software)



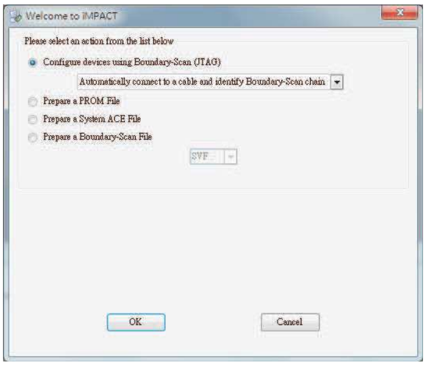
```
Launching : "Manage Configuration Project (
For further information on the status of th
Process "Manage Configuration Project (iMPA
```

In the ISE iMPACT GUI, Select File → New Project.



```
Welcome to iMPACT
iMPACT Version: 12.4
Project: C:\Xilinx\12.4\ISE_DS\\auto_project.ipf created.
```

Select Configure devices using Boundary-Scan (JTAG), click OK.



Welcome to iMPACT

Please select an action from the list below

● Configure devices using Boundary-Scan (JTAG)

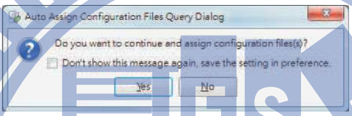
Automatically connect to a cable and identify Boundary-Scan chain

○ Prepare a PROM File

○ Prepare a System ACE File

○ Prepare a Boundary-Scan File

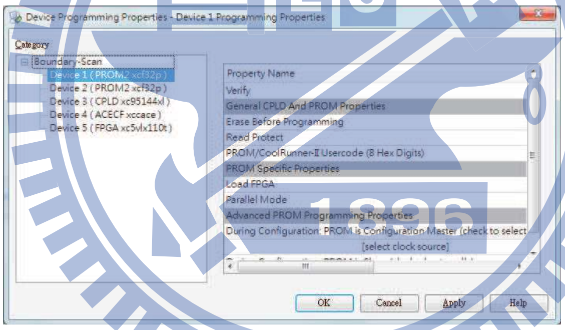
Click No and then click Cancel



Auto Assign Configuration Files Query Dialog

Do you want to continue and assign configuration files(s)?

□ Don't show this message again, save the setting in preference.



Device Programming Properties - Device 1 Programming Properties

Category

- Boundary-Scan
  - Device 1 (PROM2 xcf32p)
  - Device 2 (PROM2 xcf32p)
  - Device 3 (CPLD xc95144xl)
  - Device 4 (ACECF xccace)
  - Device 5 (FPGA xc5vlx110t)

Property Name

Verify

General CPLD And PROM Properties

Erase Before Programming

Read Protect

PROM/CoolRunner-II Usercode (8 Hex Digits)

PROM Specific Properties

Load FPGA

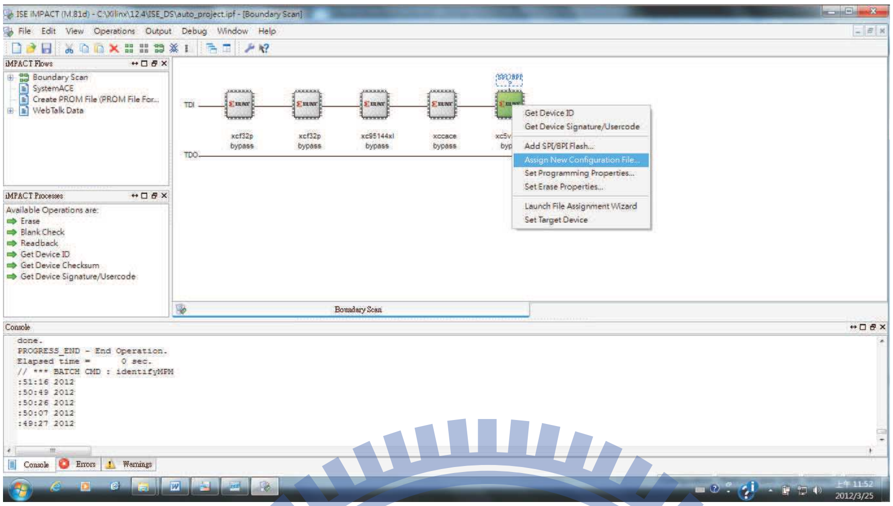
Parallel Mode

Advanced PROM Programming Properties

During Configuration: PROM is Configuration Master (check to select

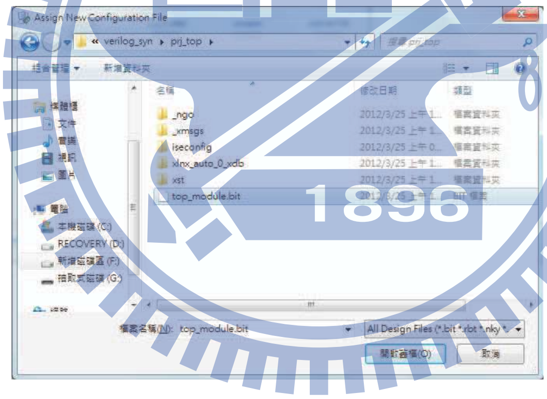
[select clock source]

Select Device: xc5vlx110t, right-click → Assign New Configuration File.


ISE iMPACT (M.81d) - C:\Xilinx\12.4\ISE_DS\auto_project.ipf - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File For...
WebTalk Data
iMPACT Processes
Available Operations are:
Erase
Blank Check
Readback
Get Device ID
Get Device Checksum
Get Device Signature/Usercode
TDI
TDO
SPI/BPI
xcf32p bypass
xcf32p bypass
xc95144xl bypass
xccace bypass
xc5v byp
Get Device ID
Get Device Signature/Usercode
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Set Target Device
Boundary Scan
Console
Errors
Warnings
上午 11:52
2012/3/25

```
done.
PROGRESS_END - End Operation.
Elapsed time =       0 sec.
// *** BATCH CMD : identifyMPM
:51:16 2012
:50:49 2012
:50:26 2012
:50:07 2012
:49:27 2012
```

Select .bit File, click Open.

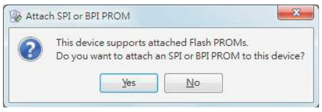


名稱	修改日期	類型
_ngo	2012/3/25 上午 1...	檔案資料夾
_xmsgs	2012/3/25 上午 1...	檔案資料夾
iseconfig	2012/3/25 上午 0...	檔案資料夾
xlnx_auto_0_xdb	2012/3/25 上午 1...	檔案資料夾
xst	2012/3/25 上午 1...	檔案資料夾
top_module.bit	2012/3/25 上午 1...	BIT 檔案

檔案名稱(N): top_module.bit

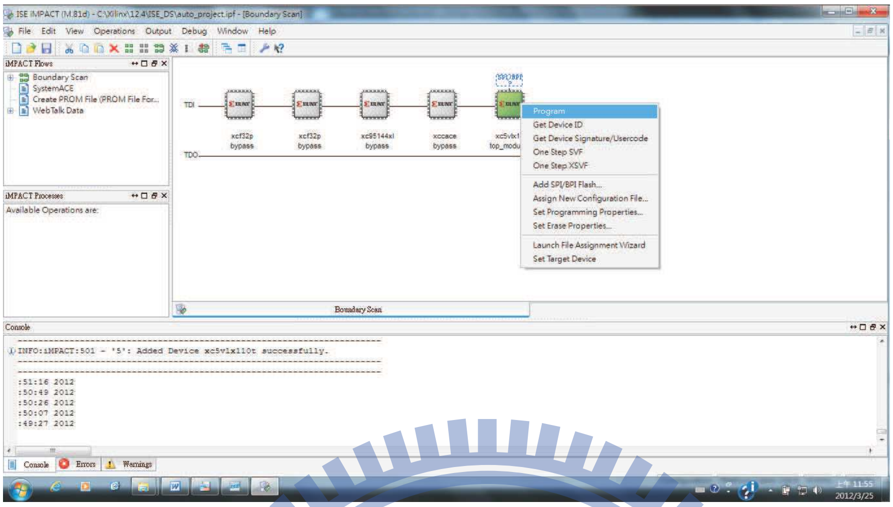
All Design Files (*.bit *.rbt *.nky *.

Click No.



This device supports attached Flash PROMs.
Do you want to attach an SPI or BPI PROM to this device?

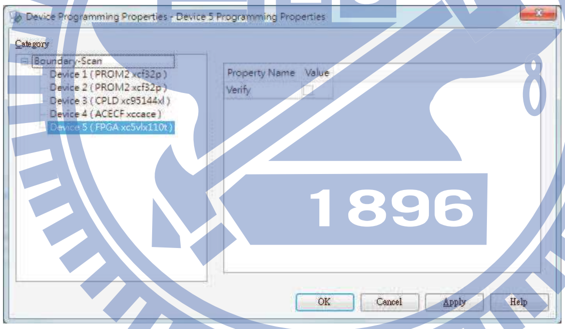
Select Device: xc5vlx110t, right-click → Program.



ISE iMPACT (M.81d) - C:\Xilinx\12.4\ISE_DS\auto_project.ipf - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File For...
WebTalk Data
iMPACT Processes
Available Operations are:
TDI
TDO
SPI/BPI
xcf32p
bypass
xcf32p
bypass
xc95144xl
bypass
xccace
bypass
xc5vlx1
top_modu
Program
Get Device ID
Get Device Signature/Usercode
One Step SVF
One Step XSVF
Add SPI/BPI Flash...
Assign New Configuration File...
Set Programming Properties...
Set Erase Properties...
Launch File Assignment Wizard
Set Target Device
Boundary Scan
Console
Errors
Warnings
上午 11:55
2012/3/25

```
--------------------------------------------------------------------------------
INFO:iMPACT:501 - '5': Added Device xc5vlx110t successfully.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
:51:16 2012
:50:49 2012
:50:26 2012
:50:07 2012
:49:27 2012
```

It will choose Devise5 and then click OK.

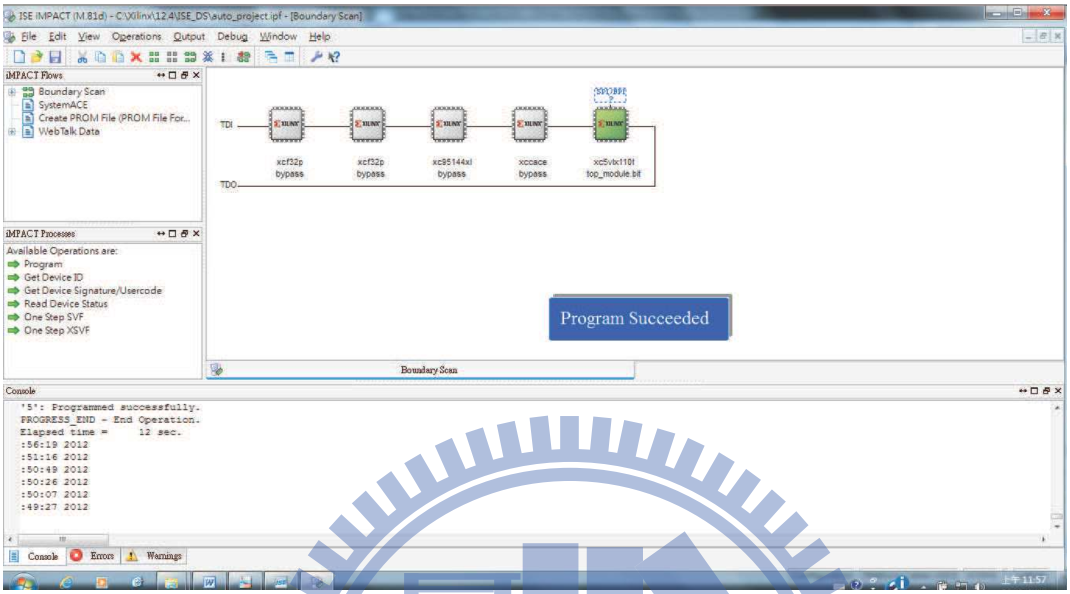


Device Programming Properties - Device 5 Programming Properties
Category
OK
Cancel
Apply
Help

Boundary-Scan
- Device 1 (PROM2 xcf32p)
- Device 2 (PROM2 xcf32p)
- Device 3 (CPLD xc95144xl)
- Device 4 (ACECF xccace)
- Device 5 (FPGA xc5vlx110t)

Property Name	Value
Verify	☐

The GUI shows Program Succeeded.

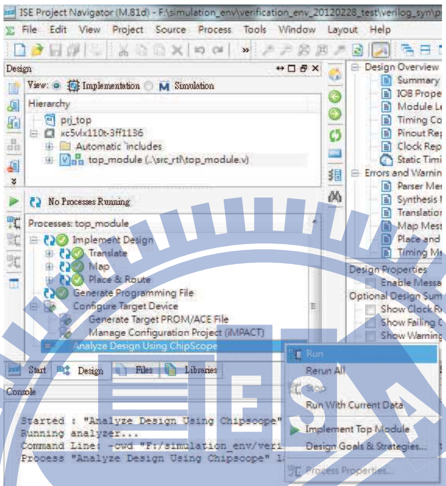


```
'5': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time =      12 sec.
:56:19 2012
:51:16 2012
:50:49 2012
:50:26 2012
:50:07 2012
:49:27 2012
```

4.4 Verification using ChipScope Pro

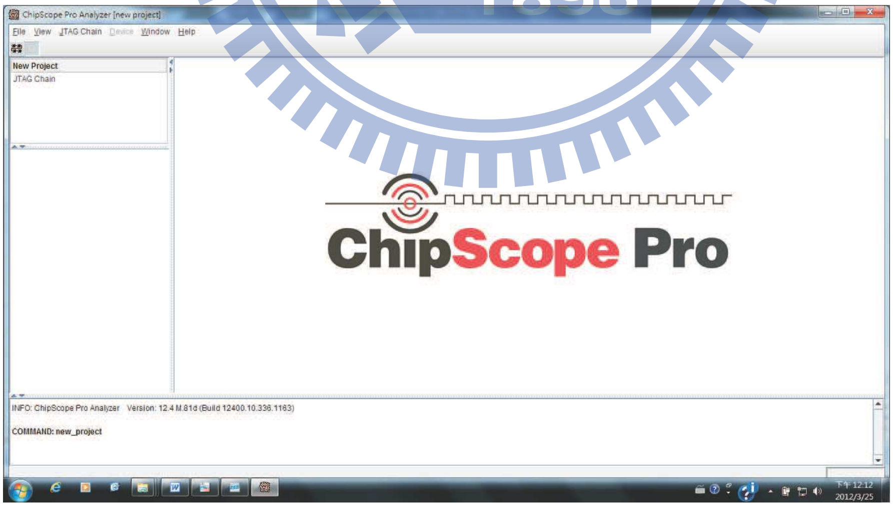
- **Running ChipScope Pro**

In the Processes Pane - Analyze Design Using ChipScope, right-click → Run (Open ChipScope)



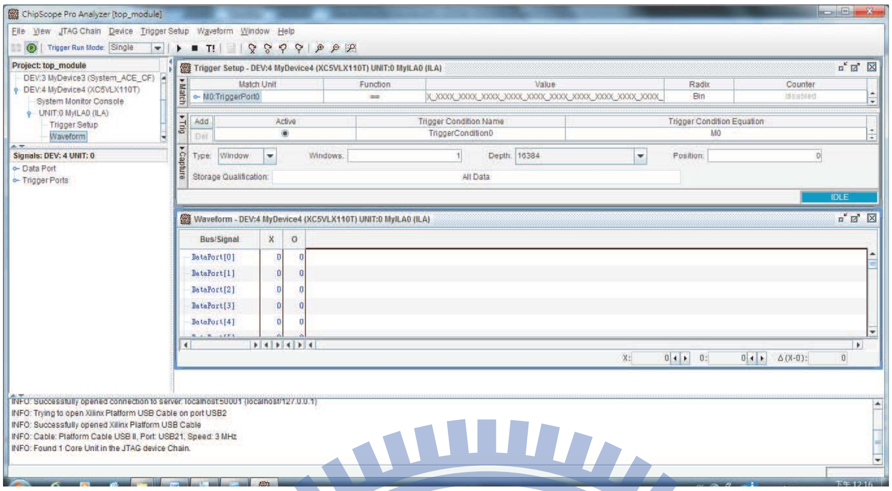
```
Started : "Analyze Design Using Chipscope"
Running analyzer...
Command Line: -cwd "F:/simulation_env/veri
Process "Analyze Design Using Chipscope" 1
```

Click Open JATA Chain button.


ChipScope Pro Analyzer [new project]
File View JTAG Chain Device Window Help
New Project
JTAG Chain
ChipScope Pro
下午 12:12
2012/3/25

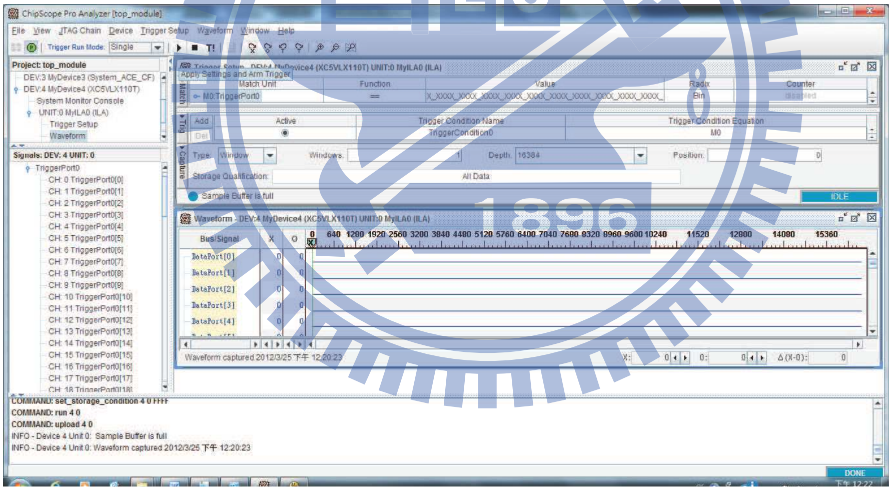
```
INFO: ChipScope Pro Analyzer   Version: 12.4 M.81d (Build 12400.10.336.1163)

COMMAND: new_project
```

We can see Project, Signals, Trigger Setup, Waveform view.



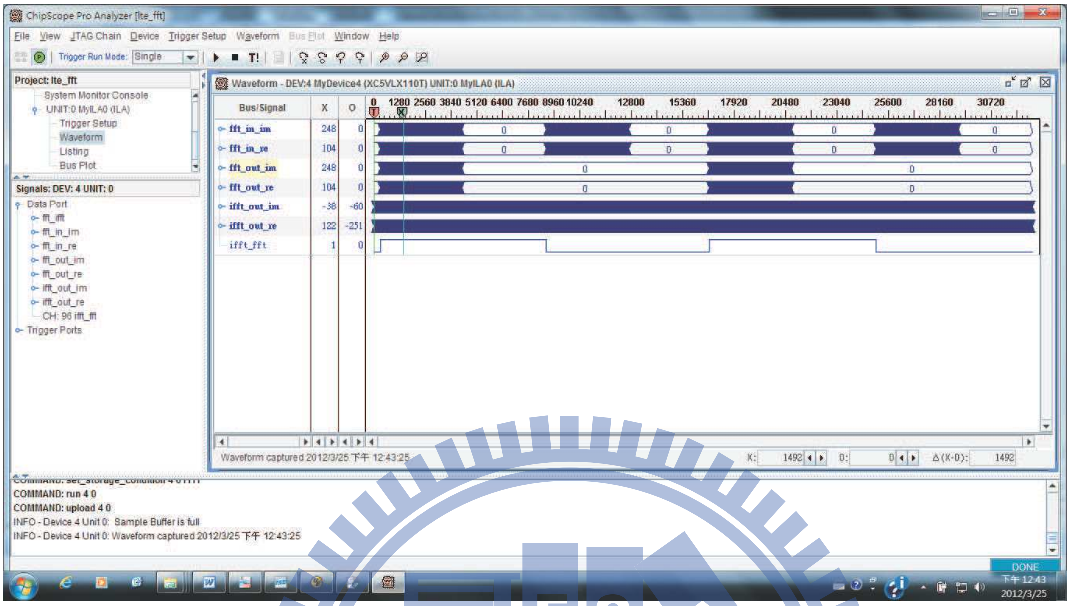
```
INFO: Successfully opened connection to server: localhost:50001 (localhost/127.0.0.1)
INFO: Trying to open Xilinx Platform USB Cable on port USB2
INFO: Successfully opened Xilinx Platform USB Cable
INFO: Cable: Platform Cable USB II, Port: USB21, Speed: 3 MHz
INFO: Found 1 Core Unit in the JTAG device Chain.
```

Click play button, you can run ChipScpoe Pro.



```
COMMAND: set_storage_condition 4 0 FFFF
COMMAND: run 4 0
COMMAND: upload 4 0
INFO - Device 4 Unit 0:  Sample Buffer is full
INFO - Device 4 Unit 0: Waveform captured 2012/3/25 下午 12:20:23
```

After editing our signal, we can get internal FPGA signal.

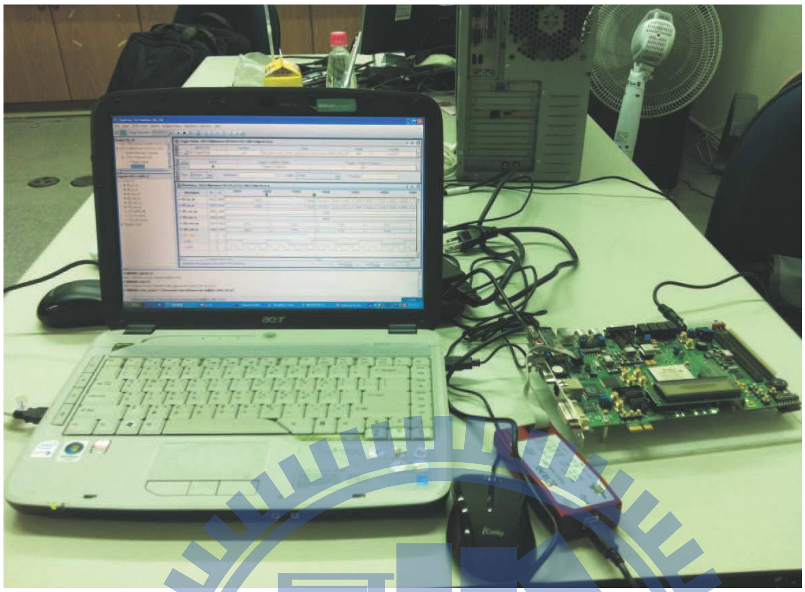


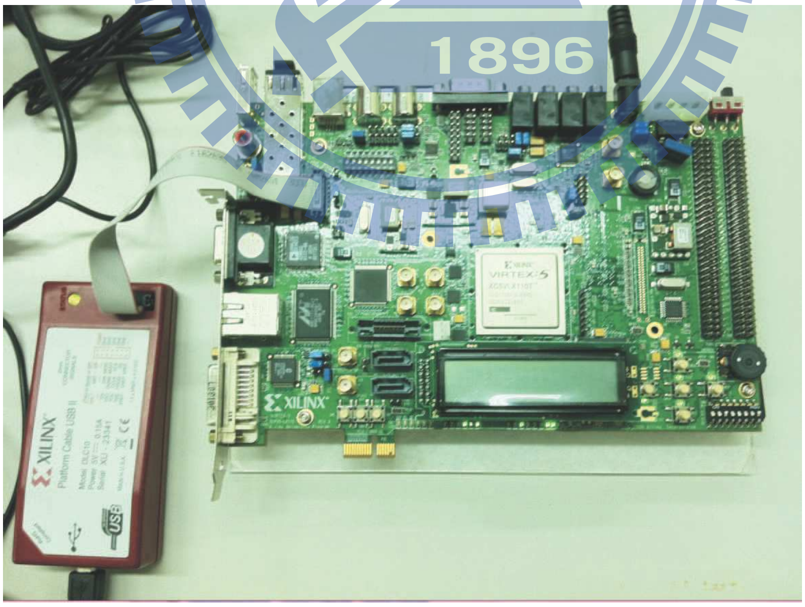
Bus/Signal	X	O
fft_in_im	248	0
fft_in_re	104	0
fft_out_im	248	0
fft_out_re	104	0
ifft_out_im	-38	-60
ifft_out_re	122	-251
ifft_fft	1	0

Waveform captured 2012/3/25 下午 12:43:25

X: 1492 O: 0 Δ(X-O): 1492

```
COMMAND: set_storage_condition 4 0 1111
COMMAND: run 4 0
COMMAND: upload 4 0
INFO - Device 4 Unit 0:  Sample Buffer is full
INFO - Device 4 Unit 0: Waveform captured 2012/3/25 下午 12:43:25
```




XILINX
VIRTEX-5
XILINX
Platform Cable USB II

Chapter 5

Conclusions

In this thesis, we proposed the variable FFT for uplink of LTE application. In chapter 2 we discuss the several FFT architectures which can be applied with various FFT size and present subcarrier mapping. In chapter 3 we use ripple-like ON/OFF stage control and early access of reordering to implement our timing plan design. In chapter 4 we discuss how to build the system platform in MATLAB and verilog and FPGA implement flow. Finally, we use FPGA to verify our design.

Bibliography

[1] Shousheng He and Mats Torkelson, "Designing Pipeline FFT Processor for OFDM (de) Modulation," URSI International Symposium on Signals, Systems and Electronics, pp. 257-262, 1998.

[2] Y. T. Lin, P. Y. Tsai, and T.D.Chiueh, "Low-power Variable-length Fast Fourier Transform Processor,"IEEE Proc.-Computer. Digit. Tech., Vol. 152, No. 4, pp 499-506, July 2005.

[3] ETSI TS 101 270-2 (v1.1.1): "Transmssiion and multiplexing (TM); access transmission systems on metallic access cables; very high speed digital subscriber line (VDSL); Part 2: Transceiver specification,"Feb. 2001.

[4] ETSI EN 300 401 (v1.3.2): "Radio broadcasting systems; digital audio broadcasting (DAB) to mobile, portable and fixed receivers,"Sep. 2000.

[5] ETSI EN 300 744 (v1.2.1): "Digital video broadcasting (DVB); framing structure, channel coding and modulation for digital terrestrial television,"Jul. 1999.

[6] Y. W. Lin and C. Y. Lee, "Design of an FFT/IFFT Processor for MIMO-OFDM Systems," IEEE Trans. Circuits Syst. I, Reg. Papers, vol.54, no.4, pp.807-815, Apr. 2007.

[7] IXIA, "SC-FDMASingle Carrier FDMA in LTE," Technical report, can be accessed $http : //www.ixiacom.com/pdfs/library/white_papers/SC - FDMA - INDD.pdf$

[8] Hyung G. Myung, Junsung Lim, and David J. Goodman, "Single Carrier FDMA for Uplink Wireless Transmission," IEEE Vehicular Technology Magazine, September 2006.

[9] ISE In-Depth Tutorial UG695 (v13.1)