MAJOR THESIS

ON

"DESIGN AND IMPLEMENTATION OF SYNCHRONIZATION AND SWITCHING COMPONENTS IN DIGITAL NETWORKS, WITH FPGA "

A dissertation submitted in partial fulfillment of the requirement for the degree of

MASTER OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION

Submitted by :

PRAVEEN KUMAR

College Roll No. 16/EC/03

University Roll No. 9113

M.E. (ELECTRONICS AND COMMUNICATION)

VIth SEMESTER

UNDER THE GUIDANCE OF

DR. ASOK BHATTACHARYYA

PROFESSOR & HEAD

ELECTRONICS AND COMMUNICATION DEPTT.

DELHI COLLEGE OF ENGINEERING

CERTIFICATE

This is to certify that the thesis entitled ***"Design and Implementation of Synchronization and Switching components, in Digital Networks, with FPGA", "*** being submitted by **Praveen Kumar** in the partial fulfillment of the requirement for the degree of Master of Engineering in Electronics and Communication in the Department of Electronics and Communication, Delhi College of Engineering, University of Delhi is a record of bonafide work done by him under my supervision and guidance. It is also certified that the dissertation has not been submitted elsewhere for any other degree.

(Prof. A. BHATTACHARYYA)
Head of Department,
Project Guide
Department of Electronics & Communication
Delhi College of Engineering
Delhi - 110042

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without a mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

I am grateful to **Prof. A. BHATTACHARYYA** (HOD, ECE Deptt.) for providing us an opportunity to undertake this project and for being my project guide for taking keen interest in my work and for his constant monitoring and invaluable guidance and support through out the course of my project. I profusely thank him for having patience to clear my doubts and channelise my efforts. His cheerful disposition made my work all the more enjoyable.

PRAVEEN KUMAR

College Roll No. 16/EC/03
University Roll No. 9113
Delhi College of Engineering
University of Delhi, Delhi.

INDEX

1. SYNCHRONIZATION

1.1 SYNCHRONIZATION INTRODUCTION

Synchronization is *the act of synchronizing* (Webster's Ninth New Collegiate Dictionary) i.e. making synchronous (cf. the Greek etymon σύγχρουος)the operation of different devices or the evolving of different processes by aligning their time scales.

Many operations in digital systems must obey a precedence relationship. If two operations obey some precedence, then synchronization ensures that operation follow in the correct order. At the hardware level, synchronization is accomplished by distributing a common timing signal to all the modules of the system. At a higher level of abstraction, software processes synchronize by exchanging messages.

Depending on the application field, different systems of *abstractions* are adopted usefully, structured in a hierarchical fashion, where each level of abstraction relies on the features of the abstraction level below and hides unnecessary details to the higher level. Abstractions enable the designers to ignore such unnecessary details and focus on essential features, thus making easier achieving a greater complexity of the system designed.

In digital hardware systems, a common approach is to structure the system representation in abstraction levels such as the *physical level*, in which the designer is concerned about physical laws governing semi conductor properties; the *circuit level*, where he deals with transistor, resistor etc.; the *element level* focused on gates, logical ports etc.; the *module level*, where elements are grouped to form more complex entities, such as memories, logic units, CPUs etc.

Whichever is the abstraction criterion adopted in describing hardware and software systems, many are the entities mutually correlated, at any level, whose correct operation relies on temporal coordination. Though, entities of different abstraction levels, both in hardware and software systems, usually require different and independent 'synchronization' functions. Elliptical term synchronization is adopted to refer to whole set of heterogeneous issues where temporal coordination is essential.

1.2 HITORICAL PERSPECTIVE

The modern telecommunication networks result from along evolution process, started since the end of the 19th century:

Network synchronization, at first an unknown issue as not relevant to network operation and performance, has played a role of increasing importance in telecommunication throughout this evolution process, especially since transmission and switching turned digital.

Transmission and switching are the two basic functions of any telecommunication network, and in particular telephone networks.

Transmission is the action of conveying information point-to-point, for example from one node in a network to another one directly linked to it by a physical channel. Moreover, transmission can also be from one point to multiple points (multicast) or even from one point to all listeners on the medium (broadcast).

Switching, on the other hand, is the function of connecting a given input-output pair in nodes where multiple transmission links are terminated. It deals thus with the dynamic assignment of the transmission channels available in a network, on the basis of user connection requests.

To make an analogy with railways, transmission systems are the tracks and switching nodes are the shunts. Transmission and switching are the complementary foundations on which all the telecommunication services are based. Both Transmission and Switching were analog first and then one after the other turned to digital technology.

The evolution of digital transmission and switching technology for the public telephones with isolated digital transmission links between analog switching machines or analog radio transmission systems. The fact that digital technology was used was transparent to the interfaces. Thus there was no need to relate internal clock rate of one system to the internal clock rate of another system.

Even as higher level multiplexing systems were developed there was no need (nor viable means) of relating the clock rates of the higher rate multiplexed signals with the clock rates of lower rate tributaries. Indeed the transmission equipment based on PDH technology does not need to be synchronized, since the bit justification technique allows the multiplexing of asynchronous tributaries with substantial frequency offsets.

Problems began to arise wit such asynchronous architecture when digital technology moved to switching machines too. Digital switching equipment requires to be synchronized in order to avoid slips at input elastic stores. And while slips do not significantly affect normal phone conversations, they may be troublesome indeed on some data services! The introduction of circuit switched data networks and of ISDN, therefore, yielded first the need for more stringent synchronization requirements.

The ongoing spread of SDH/SONET technology in transmission network has really made synchronization a hot topic in standard bodies since 1990s.The need for adequate network synchronization facilities has become more and more stringent in order to fully exploit SDH/SONET capabilities; it is widely recognized that SDH/SONET transmission may rely on a suitable dependable timing distribution to fully meet all its benefits, in particular because pointer action may yield excess jitter on transported tributaries.

Beyond SDH/SONET needs, anyway, nowadays network synchronization facilities are unanimously considered as profitable network resource, allowing slip free digital switching, enhancing the performance of ATM based transport services and serviceable for improving the quality of a variety of services (e.g. ISDN, mobile cellular telephony, etc.). For this reason, most major network operators have set up national synchronization networks, in order to distribute a common timing reference to each node of the telecommunication network, On the standardization side, ITU-T and ETSI bodies released new synchronization standards, suitable for operation of modern digital communication network, specifying more stringent and complex requirements for jitter and wander at synchronization interfaces, for clock accuracy and stability and for the synchronization network architecture.

Most modern synchronization networks are provided with management systems. The main management functions relevant to synchronization network management lie in the areas of fault, configuration, performance and security management.

Most advanced synchronization networks are provided with monitoring systems that allows to verify continuously, in real time, the performance achieved in timing distribution. The rationale of synchronization performance monitoring is the need to be proactive, i.e. to detect timing degradations well before they impact service.

1.3 SYNCHRONIZATION IN TELECOMMUNICATION

The term synchronization is familiar in a somewhat restricted sense , meaning only acquisition and tracking of a clock in a receiver, with reference to the periodic timing information contained in the receive signal. More properly speaking this should be referred to as carrier or symbol synchronization. On the contrary, synchronization plays an essential role in several other areas in telecommunications, at different level of abstraction and in different context too.

At different abstraction levels, the main contexts in which the word synchronization is used in telecommunication are the following:

- *Carrier synchronization*, i.e. the extraction of the carrier from a modulated signal in coherent demodulation;
- *Symbol synchronization*, i.e. the identification of sampling and decision times in digital demodulation, in order to extract the logical information from the received analog signal;
- *Word and frame synchronization*, i.e. the identification of start and end of code words or of group of code words(frames), or also the delineation of the frames in the raw and undifferentiated stream of received bits;

- *Packet synchronization*, i.e. the delay equalization of packet arrival times in order to reconstruct a user circuit with constant bit rate over a packet switched network;
- *Network synchronization*, i.e. the distribution of a common timing over a network of clocks, spread over an even wider geographical area;
- *Multimedia synchronization*, i.e. the orchestration of heterogeneous elements (images, text, audio, video, etc.) in a multimedia communication at different (e.g. physical and human interface) levels of integration;
- *Synchronization of real-time clocks*, i.e. a substantially different kind of network synchronization in which the distribution of the absolute time (e.g. the national standard time) across a telecommunication network is concerned, mainly to network management purposes.

1.4 NETWORK SYNCHRONIZATION ARCHITECTURES

Network synchronization is a comprehensive expression that addresses in a wide sense any distribution of time and frequency over a network of clocks. Its goal may be either

1. To align the absolute time scales of network nodes, thus aiming for instance at aligning local clocks to the Universal Time Coordinated (UTC)
2. To align the timing signals (or more precisely, their significant instants) generated by local clocks, independently from a constant phase offset among them. Thus aiming at minimizing phase fluctuations around such average phase offset (example : synchronization of synchronous digital multiplexers or digital switching exchanges in order to avoid slips at input elastic stores);
3. To equalize the frequencies of local clocks, without controlling their phase relationship (example: the distribution of a standard signal to PLL based slave clocks).

In a network synchronized as in case (1), local timing signals are synchronous and their total phases are aligned. Therefore, this network synchronization requires estimation and compensation of transmission delays on synchronization signals directed to each node.

In a network synchronized as in case (2), local timing signals are synchronous but there is no need to estimate transmission average delays of synchronization signals.

In a network synchronized as in case (3), finally, timing signals are just mesochronous.

In most cases network synchronization is intended as in case (2) and is achieved by transferring chrono signals (i.e.some pseudo-periodic signals such as sine or square waves), which carry a timing information with the uncertainty of the integer number of periods elapsed since the signal was generated (total transmission delay).

A synchronization network is the facility implementing network synchronization. It is able to provide all telecommunication networks with reference timing signals of required quality. Most modern telecommunication operators have set up one synchronization network to synchronize their switching and transmission networks.

Basic elements of synchronization network are the nodes (autonomous and slave clocks) and links interconnecting them. An autonomous clock is a stand-alone device able to generate a timing signal, starting from some periodic physical phenomenon. A slave clock on the other hand, generates a timing signal having phase locked to a reference timing signal at its input. Slave clocks are usually implemented as PLLs (Phase Locked Loops).

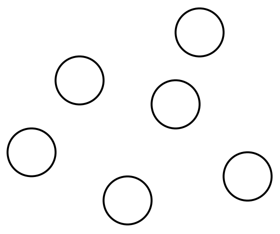
Time and frequency are distributed by using the communication capacity of the links interconnecting the clocks (e.g. copper cables, optical fibres, radio links). However, network nodes may be many and spread over a wide geographical area. Therefore two distinct issues must be faced:

- How to transfer timing from one node to the other (the tactics of point-to-point timing transfer);
- How to organize timing distribution among all nodes of the network (the strategy of network synchronization).

1.5 NETWORK SYNCHRONIZATION STRATEGIES

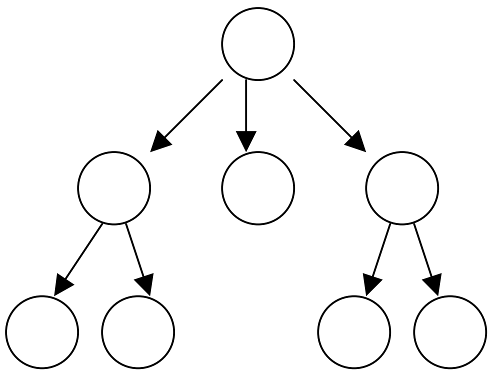
Network synchronization plays a central role in modern digital telecommunications, determining the quality of most services offered by network provider to its customers. To this purpose many different network synchronization strategies have been conceived. Among them following three have found wide application throughout the last decades: full plesiochrony, hierarchical master-slave (HMS) synchronization, and mutual synchronization. The main feature of these strategies are as follows.

Full Plesiochrony (Anarchy):



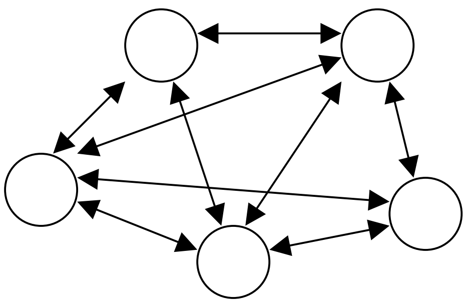
It is actually no synchronization strategy (i.e., it does not involve any synchronization distribution).Each network node is equipped with an independent clock. Anarchy is the easiest form of government, but it relies on good behavior of the single elements. Due to lack of any timing distribution, the synchronization of the operation of different nodes is entrusted to accuracy of network clocks, which therefore must feature excellent performance.

Hierarchical Master –Slave Synchronization (Depotism):



The principal of master slave strategies is based on the distribution of the timing reference from a clock (master) to all other clocks of the network (slaves), directly or indirectly. Depotism is generally considered unethical, but it is certainly effective in ensuring very tight control of the slaves: an MS network is synchronous with the master clock and stable by definition. The HMS strategy is currently the most widely adopted to synchronize modern digital telecommunication networks, due to the excellent timing performance and reliability that can be achieved at limited cost.

Mutual Synchronization (Democracy):



Mutual synchronization is based on direct mutual control among the clocks so that output frequency of each is the result of the “suggestions” of the others. Such a pure democracy looks appealing: there are no masters and no slaves, but mutual cooperation. However, the behavior of the mutually controlled elements is hard to govern. Modeling the behavior of such networks, or even ensuring the stability of the control algorithms, can be very complex task. Network so designed thus tend to be quite expensive, but extremely reliable. Therefore, until now the field of application of mutual synchronization has been mostly limited to special cases (e.g., military networks).

Timing Relation ship between digital signals:

Isochronous : Digital signal in which time intervals between significant instants have, at least on the average, the same duration or durations which are integer multiples of shortest one.
Two *synchronous* digital signals are isochronous digital signals whose respective timing signal have the same frequency, at least on the average, and a phase relationship controlled precisely.
Two *mesochronous* digital signals are isochronous, asynchronous digital signals, whose respective timing signals have the same frequency, at least on the average, but no control on phase relationship.
Two *plesiochronous* digital signals are isochronous, asynchronous digital signals, whose respective timing signals have the same frequency values only nominally, but actually different within a given tolerance range.
Two *heterochronous* digital signals are isochronous, asynchronous digital signals, whose respective timing signals have different nominal frequencies.

To give sound examples of above abstract concepts, a locked Phase Locked Loop (PLL) outputs a timing signal which is synchronous with the input signal, owing to the feedback control on the phase error between them. A Frequency Locked Loop (FLL), i.e., a feedback system operating like a PLL but instead controlling the frequency error between the input and the output signals, outputs a signal which is mesochronous with the input. Two oscillators, even if designed and built as equal by the same supplier, output two plesiochronous timing signals, owing to unavoidable manufacturing tolerances. Finally, two digital signals with different rates (e.g., 2.048 Mb/s and 8.448Mb/s signals) are heterochronous.

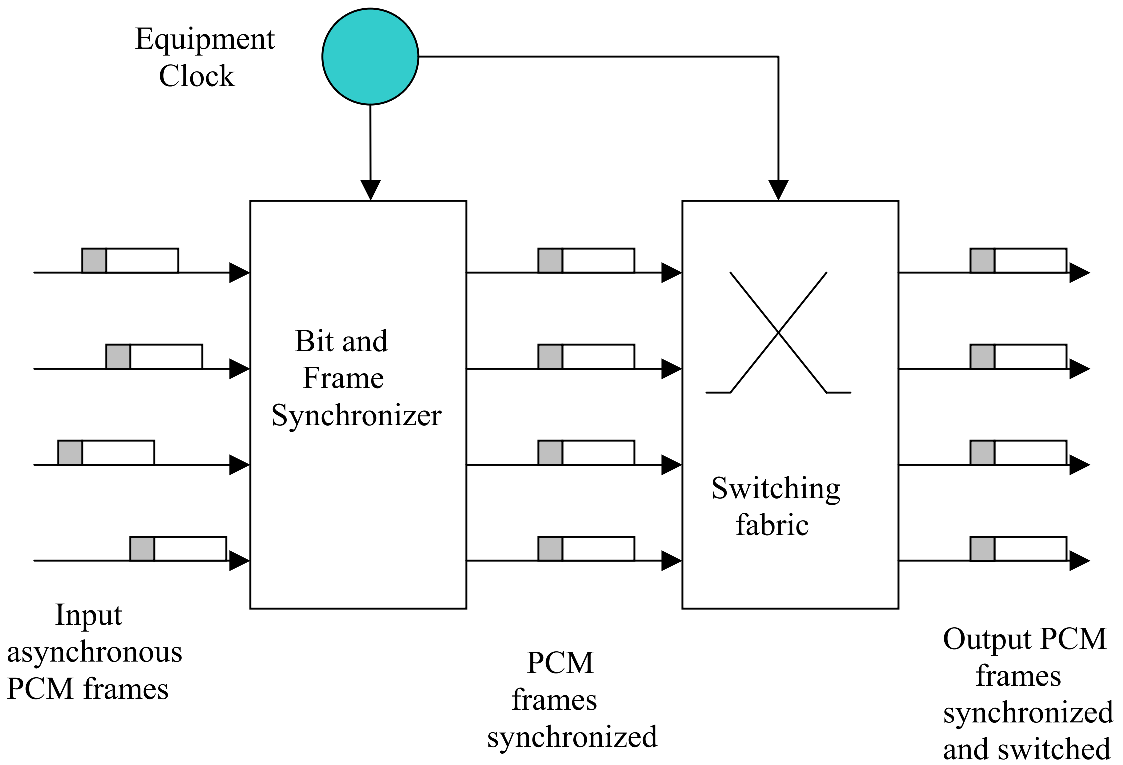
1.6 SYNCHRONISATION AND DIGITAL SWITCHING

The advent of digital TDM techniques yielded a progressive integration of transmission and switching, since the PCM primary multiplex frame structure allows exploiting of the TDM principle for digital switching of circuit connections as well.

Digital Switching Requires Time Alignment Of The Input PCM Frames:

The European 2.048 Mb/s PCM frame is made of 32 octets (time slots), 30 of which carry single 64 Kb/s telephone channels, while the North American 1.544 Mb/s PCM frame is made of 24 slots. Digital switching is based on moving octets (speech samples) from one time slot to another, from one input signal to another output signal. Time slot exchanging is basically done by delaying, by a suitable time interval, the incoming octets before retransmitting them in the output frame at the right place (time).

It clearly appears that digital switching can take place only if incoming frames (asynchronous since they can be generated by different pieces of equipment with different clocks) are made synchronous, with frame starts aligned, so that correspondent time slots at different inputs are perfectly time aligned. Therefore, one of the tasks of the input line units of a digital switching exchange is to synchronize bits and frames of incoming PCM signals before feeding them into the switching fabric, as outlined in figure. In this figure, for the sake of simplicity, only one frame per line is depicted (with alignment word shaded), and the time slot interchanging in the PCM frames is not pointed out.


Equipment Clock
Bit and Frame Synchronizer
Switching fabric
Input asynchronous PCM frames
PCM frames synchronized
Output PCM frames synchronized and switched

Thus, for synchronization to be achieved Phase Locked Loops (PLLs) are to be used and hence are vital components for any synchronized network.

The main task of these PLLs is, on the one hand, to ensure adequate short-term stability by filtering phase fluctuations accumulated by pilots along the transmission links, and on the other to provide in any case an output reference frequency, even under loss of input pilot, by *free running* operation of the local oscillator. Free-run frequency accuracy requested to limit distortion in the demodulated signals is in order of 10^{-7}. Such a frequency accuracy is enough to ensure an adequate transmission quality of telephone channels, even under pilot frequency losses lasting for the mean time for restoring.

2. PHASE LOCKED LOOPS

2.1 PHASE LOCKED LOOPS

A phase-locked loop (PLL) is a circuit which causes a particular system to track with another one. More precisely, a PLL is a circuit synchronizing an output signal (generated by an oscillator) wit a reference or input signal in frequency as well as in phase. In the synchronized –often called *locked* – state the phase error between the oscillator's output signal and the reference signal is zero, or very small.

If a phase error builds up, a control mechanism acts on the oscillator in such a way that the phase error is again reduced to minimum. In such a control system the phase of output signal is actually locked to the phase of input signal. This is why it is referred to as *phase-locked loop*.

2.2 TYPES OF PLL

1. Linear PLL (LPLL)
2. Digital PLL (DPLL)
3. All Digital PLL (ADPLL)
4. Software PLL (SPLL)

2.3 FUNDAMENTAL BLOCKS

The PLL consists of three basic fundamental blocks:

1. A Phase Detector (PD)
2. A voltage controlled oscillator (VCO)
3. A loop filter (LF)

The signals of interest within the PLL circuit are defined as follows:

- The reference (or input) signal $u_1(t)$
- The angular frequency ω_1 of the reference signal

- The output signal $u_2(t)$ of the VCO
- The angular frequency ω_2 of the output signal
- The output signal $u_d(t)$ of the phase detector
- The output signal $u_f(t)$ of the loop filter
- The phase error θ_e , defined as the phase difference between signals $u_1(t)$ and $u_2(t)$

2.4 BRIEF HISTORY

The very first (PLL) were implemented as early as 1932 by de Bellesize; this French engineer is considered inventor the “coherent communication”. The PLL found broad industrial applications only when it became available as an integrated circuit. The first PLL IC’s appeared around 1965 and were purely analog devices.
An analog multiplier (four quadrant multiplier) was used as phase detector, the loop filter was build from a passive or RC filter and the well known voltage controlled oscillator was used to generate the output signal of PLL. This type of PLL is referred to as linear PLL today. In the following years the PLL drifted slowly but steadily into digital territory. The very first digital PLL (DPLL) which appeared around 1970, was in effect a hybrid device; only the phase detector was build from digital circuit, e.g., from an EXOR gate or a JK flip flop, but the remaining blocks still were analog. A few years later the “all-digital PLL (ADPLL)” was invented. The ADPLL is exclusively build from digital functional blocks hence doesn’t contain any passive components like resistors and capacitors. In analogy to filters, PLL’s can also be implemented “by software”. In this case the function of PLL is no longer performed by a piece of specialized hardware, but rather by a computer program. This last type if PLL is referred to as SPLL.
Unfortunately, LPLLs, DPLLs, and ADPLLs behave differently, so there is no common theory which covers all of these types. Consequently we must treat the various types of PLLs separately.

2.5 PLL APPLICATIONS

A Sample of PLL Applications :
The reason that PLLs are so ubiquitous is that they are so useful in so many applications.

• Carrier Recovery
• Clock/Data Recovery
• Frequency Synthesis
• Modulation/Demodulation

• PLL Applications in Control Problems

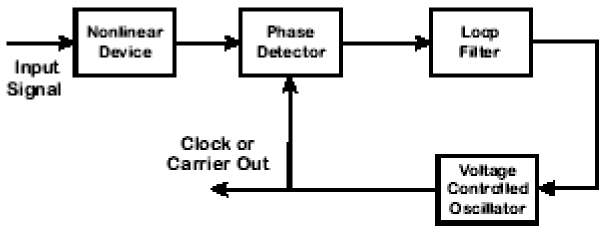
-- Disk Drive Control

-- Harmonic Compensation

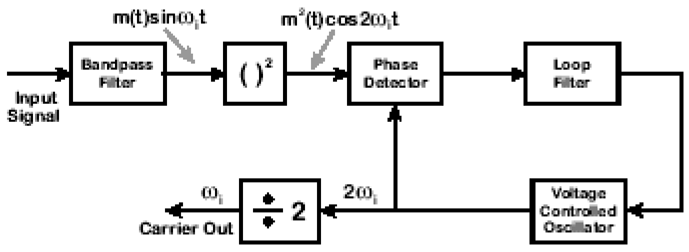
-- Motor Control

EXAMPLES :

1. Carrier Recovery

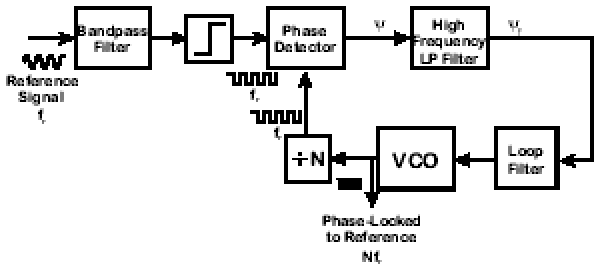


General block diagram of frequency recovery from a modulated signal.



Squaring loop to recover carrier from a BPSK modulated signal.

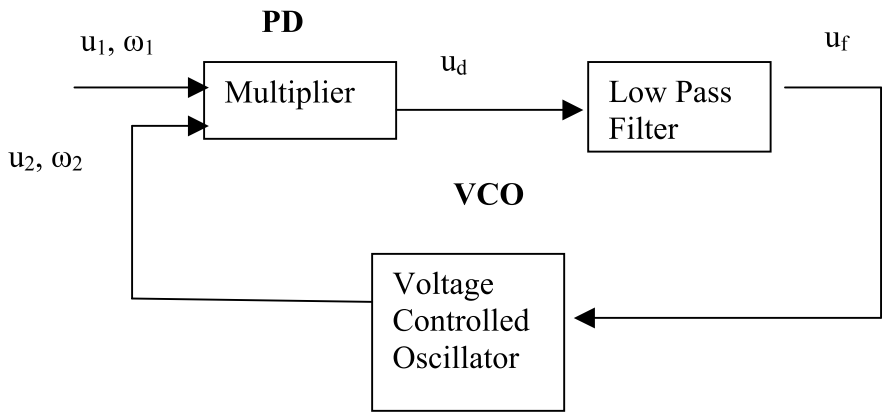
2. Frequency Synthesis



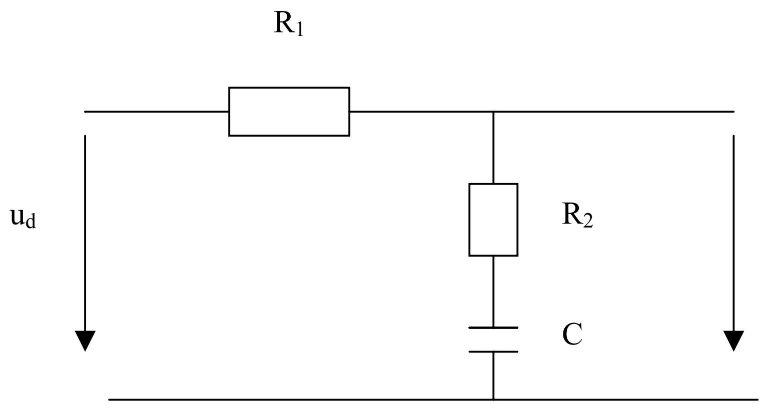
To lock a clock with an input signal of a different frequency synthesize a clock frequency from a lower frequency input. Harmonic locking loop generates a clock at N times input frequency (non-integer N is possible). Example from storage industry is DVD+RW format uses a high frequency wobble embedded in the groove walls to synthesize a write clock frequency.

2.6 THE LINEAR PHASE LOCKED LOOP (LPLL)

2.6.1 Building Blocks of LPLL



In linear PLLs, the fourth quadrant multiplier is used as phase detector. In most cases the input signal $u_l(t)$ is a sine wave with angular frequency ω_1, whereas output signal $u_2(t)$ is a symmetrical square wave with angular frequency ω_2, In the locked state the two frequencies are equal. The output signal $u_d(t)$ of the phase detector then consist of a number of terms; the first of these is a "dc" component and is roughly proportional to the phase error θ_e; the remaining terms are "ac" components having frequencies of 2 ω_1, 4 ω_1 …Because these higher frequencies are unwanted signals, they are filtered out by the loop filter (which is a low pass filter). Fig below shows a passive lag filter having one pole and one zero.

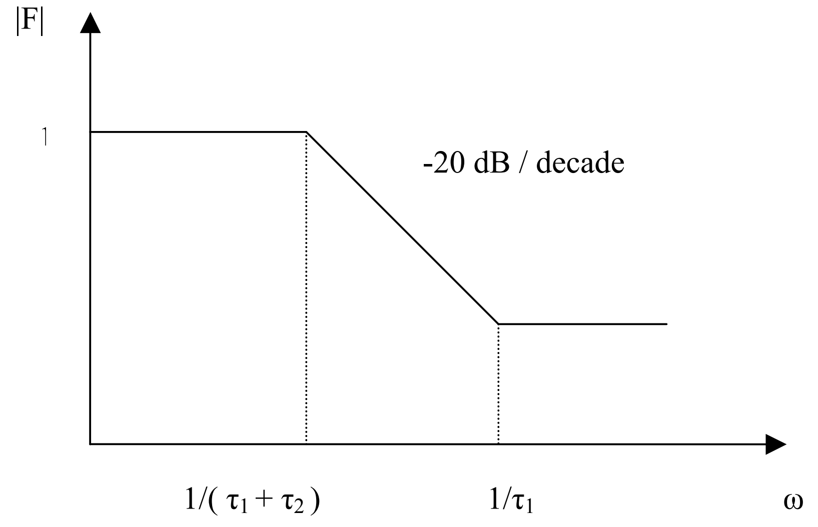


Its transfer function is given by

$F(s) = 1 + s\,\tau_2 /\ 1 + s(\tau_1 + \tau_2)$

Where $\tau_1 = R_1C$ and $\tau_2 = R_2C$

Its amplitude response is shown in figure below:



Higher order low-pass filters could be used instead of simple one-pole filters; this is done in some applications. Because each additional filter pole introduces phase shift, it is much more difficult to maintain stability in higher order systems.

2.6.2 LPLL performance in Locked State

Assuming PLL to be locked and remain locked in near future a mathematical model can be developed for phase transfer function H(s) which relates the phase θ_1 of the input signal to the phase θ_2 of the output signal.

$H(s) = \theta_2(s) / \theta_1(s)$

Where $\theta_1(s)$ and $\theta_2(s)$ are the Laplace Transforms of the phase signals $\theta_1(s)$ and $\theta_2(s)$ respectively.

To get an expression for H(s) we must know the transfer functions of the individual blocks in block diagram of LPLL. Let us start with phase detector.

Input signal of an LPLL is usually a sine wave.

$u_1(t) = U_{10} \sin(\omega_1 t + \theta_1)$

whereas output signal is usually a square wave and can therefore be written as a Walsh function

$u_2(t) = U_{20}\, \omega(\omega_2 t + \theta_2)$

The output signal of the four quadrant multiplier is obtained by multiplying the signals u_1 and u_2.

To simplify the analysis the Walsh function is replaced by its fourier series.

For $u_2(t)$ we then get

$u_2(t) = U_{20} [\, 4/\pi \cos(\omega_2 t + \theta_2) + 4/3\pi \cos(3\omega_2 t + \theta_2) + \dots \,]$

First term in square bracket is fundamental component the remaining terms are odd harmonics. For output signal $u_2(t)$ therefore we get

$u_d(t) = u_1(t)\, u_2(t) = U_{10} U_{20} \sin(\omega_1 t + \theta_1)$

$x\ [\,4/\pi \cos(\omega_2 t + \theta_2) + 4/3\pi \cos(3\omega_2 t + \theta_2) + \ldots\,]$

When PLL is locked, the frequencies ω_1 and ω_2 are identical and $u_d(t)$ become

$u_d(t) = U_{10}\, U_{20}\, [\,2/\pi \sin \theta_e) + \ldots\,]$

where $\theta_e = \theta_1 - \theta_2$ is the phase error. The first term in the series is the wanted "dc" term, whereas the higher frequency terms will be eliminated by the loop filter. Setting

$K_d = 2\, U_{10}\, U_{20} / \pi$ and neglecting higher frequency terms we get

$u_d(t) = K_d \sin \theta_e$

where K_d is called detector gain. When the phase error is small, the sine function can be replaced by its argument, and we have

$u_d(t) \approx K_d\, \theta_e$

In locked state of LPLL the Phase Detector represents a zero order block having a gain of K_d. Transfer function of a Passive Lag Filter having one pole and one zero is

$F(s) = 1 + s\,\tau_2 / \ 1 + s(\tau_1 + \tau_2)$

Where $\tau_1 = R_1C$ and $\tau_2 = R_2C$

Angular frequency of VCO is given by

$\omega_2(t) = \omega_0 + \Delta\omega_2(t) = \omega_0 + K_0\, u_f(t)$

where K_0 is called VCO gain (dimension : rad $s^{-1}V^{-1}$)

The model should yield the output phase θ_2 and not the output frequency ω_2.

By definition

$\theta_2\ (t) = \int \Delta\omega_2\, dt = K_0 \int u_f\, dt$

Laplace transform

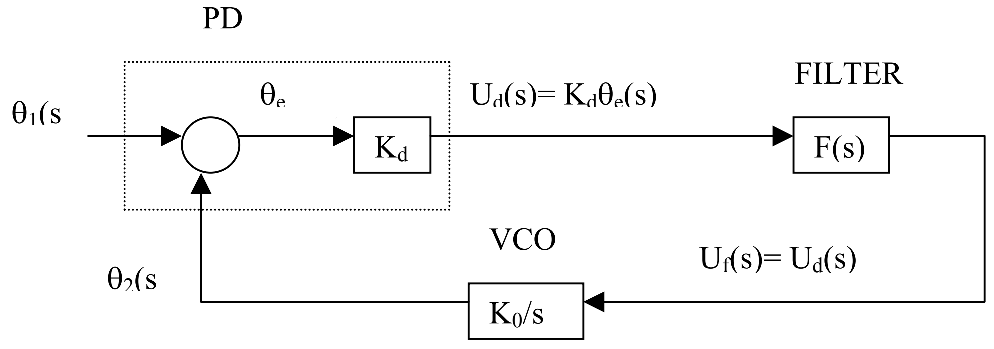
$\theta_2\ (s) = (K_0/s)\, U_f\,(s)$

The transfer function of the VCO is given by

$\theta_2\ (s) / U_f\,(s) = K_0/s$

For phase signals the VCO simply represents an integrator.

From above equations a simplified linear model of a LPLL can be drawn :



From the model Phase Transfer Function H(s) is computed. We get

$H(s) = \theta_2(s) / \theta_1(s) = K_0 K_d F(s) / s + K_0K_dF(s)$

In addition to the phase transfer function, an error transfer function $H_e(s)$ is defined by

$H_e(s) = \theta_e(s) / \theta_1(s) = s / s + K_0K_dF(s)$

Replacing F(s) by LPF transfer function we get, for passive lag filter :

$$H(s) = \frac{(K_0K_d)(1 + s\tau_2 / 1 + s(\tau_1 + \tau_2))}{s^2 + s(1 + K_0 K_d \tau_2)/(\tau_1 + \tau_2) + (K_0 K_d)/(\tau_1 + \tau_2)}$$

writing the denominator in normalized form i.e.,

Denominator $= s^2 + 2\xi\,\omega_n s + \omega_n^2$

Where ω_n is natural frequency and ξ is the damping factor.

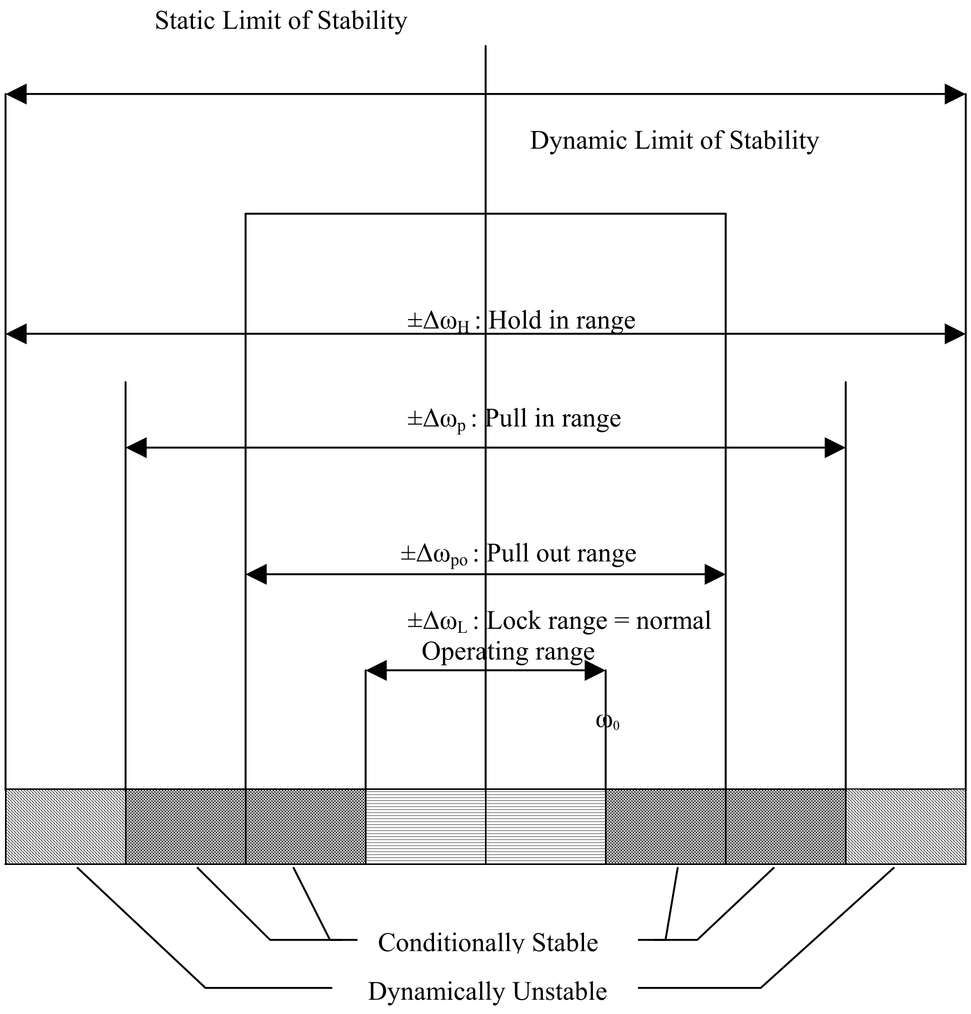
For passive lag filter :

$\omega_n = \sqrt{(K_0 K_d)/(\tau_1 + \tau_2)}$

$\xi = \omega_n/2\,(\tau_2 + 1/K_0 K_d)$

$K_0 K_d$: Loop Gain

2.7 KEY PARAMETERS OF THE LPLL


Static Limit of Stability
Dynamic Limit of Stability
$\pm\Delta\omega_H$: Hold in range
$\pm\Delta\omega_p$: Pull in range
$\pm\Delta\omega_{po}$: Pull out range
$\pm\Delta\omega_L$: Lock range = normal Operating range
ω_0
Conditionally Stable
Dynamically Unstable

Hold range, $\Delta\omega_H$: LLPL can statically maintain phase tracking, conditionally stable.

Pull out range, $\Delta\omega_{po}$	: Dynamic limit for stability, normally locked again.
Pull in range, $\Delta\omega_{p}$	: Lock again but process is slow.
Lock range, $\Delta\omega_{L}$	: Lock within single beat note between reference frequency & output frequency.

2.7.1 The Hold Range

It is the frequency range in which a PLL is able to maintain the lock statically. The LPLL locks out for ever when the frequency of the input signal exceeds the hold range.

Magnitude is calculated by frequency offset at reference input which causes phase error θ_e of $\pi/2$.

$$\omega_1 = \omega_0 + \Delta\,\omega_H$$

where is the hold range. For the phase signal we get

$$\theta_1(t) = \Delta\,\omega_H\, t$$

Laplace transform

$\theta_1(s) = \Delta\,\omega / s^2$

Phase error :

$\theta_e(s) = \theta_1(s)\, H_e(s) = (\Delta\,\omega / s^2)/\,(s / s + K_d K_0 F(s))$

using final value theorem, final phase error in time domain :

$$\lim_{t \to \infty} \theta_e(t) = \lim_{s \to 0} s\theta_e(s) = \Delta\omega / K_d K_0 F(0)$$

this is for small values of θ_e only, for greater values :

$$\lim_{t \to \infty} \sin\theta_e(t) = \Delta\omega_H / K_d K_0 F(0)$$

at the limit of hold range $\theta_e = \pi/2$ $\sin\theta_e = 1$, therefore expression for hold range :

$\Delta\omega_H = K_d K_0 F(0)$

for passive lag filter :

$\Delta\omega_H = K_d K_0$

2.7.2 The Lock Range

Assume that the LPLL is initially not locked and that the reference frequency is

$\omega_1 = \omega_0 + \Delta\,\omega$. The reference signal of the LPLL is then given by

$u_1(t) = U_{10} \sin(\omega_0 t + \Delta\omega t)$ and the output signal by

$u_2(t) = U_{20}\, \omega\, (\omega_0 t)$

Phase Detector will deliver an output signal given by

$u_d(t) = K_d \sin(\Delta\omega t) +$ higher frequency terms

The higher frequency terms can be filtered out by loop filter. The output of loop filter can be written as

$u_f(t) \approx K_d |F(\Delta\omega)| \sin(\Delta\omega t)$

this is an AC signal causing frequency modulation of the VCO. The peak frequency deviation is equal to $K_d K_0 |F(\Delta\omega)|$.

If this frequency deviation is less than the frequency offset $\Delta\omega$ then lock in process will either not take place or is at least very slow. When frequency deviation is large so that ω_2 exactly meets the reference frequency ω_1 , PLL locks within the single beat note between the reference and the output frequencies.

The condition for locking is therefore

$K_d K_0 |F(\Delta\omega)| \geq \Delta\omega$

Lock Range itself is given by

$\Delta\omega_L = K_d K_0 |F(\Delta\omega_L)|$

Approximating that lock range is much greater then corner frequencies $1/\tau_1$ and $1/\tau_2$ of the loop filter and also assuming τ_2 to be much smaller than τ_1 we can use

$F(\Delta\omega_L)| \approx \tau_2 / \tau_1$ for passive lag filter

Assuming high gain loops and making substitutions we get

$\Delta\omega_L \approx 2\zeta\omega_n$ --------- for all types of loop filters

When LPLL locks quickly ($\zeta < 1$, damped oscillations, transients die out in one clock cycle), it is reasonable to state lock-in time as

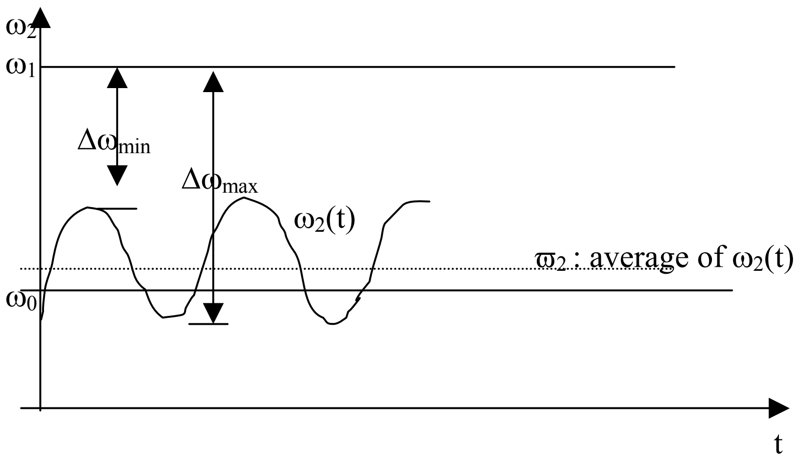
$T_L \approx 2\pi/\omega_n$ (settling time).

2.7.3 Pull in Range $\Delta\omega_p$

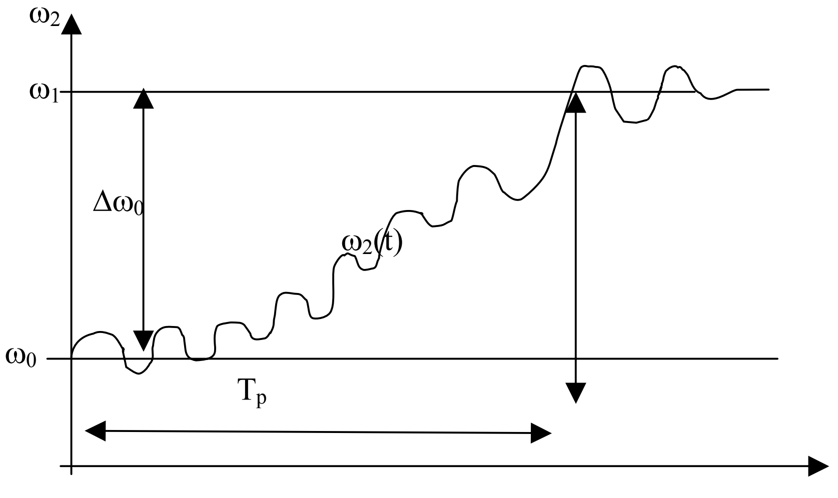
Assume again that LPLL is not locked initially, that the frequency of the reference signal is $\omega_1 = \omega_0 + \Delta\omega$, and the VCO operates at the centre frequency ω_0. Consequently the output signal u_d of the phase detector is a sine wave having the frequency $\Delta\omega$.

The difference $\Delta\omega$ between reference frequency ω_1 and output frequency $\omega_2(t)$ is not a constant; it is also varied by the frequency modulation of the VCO output signal. If the frequency $\omega_2(t)$ is modulated in the positive direction, the difference $\Delta\omega$ becomes smaller and reaches some minimum value $\Delta\omega_{min}$; if $\omega_2(t)$ is modulated in negative direction , however, $\Delta\omega$ becomes greater and reaches some maximum value $\Delta\omega_{max}$, Because $\Delta\omega(t)$ is not a constant, the VCO frequency is modulated non harmonically, that is, the duration of the half-period in which $\omega_2(t)$ is modulated in the positive direction becomes *longer* than that of the half-period in which $\omega_2(t)$ is modulated in the negative sense. As a consequence the average frequency ϖ_2 of the VCO is now higher than it was without any modulation, i.e., the VCO frequency is pulled in the direction of the reference signal.

The asymmetry of the waveform $\omega_2(t)$ is greatly dependent on the value of the average offset $\Delta\omega$; the asymmetry becomes more marked as $\Delta\omega$ is decreased. If the avereage value of $\omega_2(t)$ is pulled somewhat in the direction of ω_1 (which is assumed to be greater than ϖ_2), the asymmetry of the $\omega_2(t)$ waveform becomes stronger. This in turn causes ϖ_2 to be pulled even more in the positive direction. This process is regenerative under certain conditions, so that the output frequency ω_2 finally reaches the reference frequency ω_1 . This phenomenon is called the *pull-in* process. Mathematical analysis shows that a pull-in process occurs whenever the initial frequency offset $\Delta\omega$ is smaller than a critical value, the *pull-in range* $\Delta\omega_p$. If , on the other hand, the initial frequency offset $\Delta\omega$ is larger than $\Delta\omega_p$, a pull-in process does not take place because the pulling effect is not then regenerative.



In the unlocked state of the PLL the frequency modulation of the VCO output signal is non harmonic . This causes the average value of the VCO output frequency to be pulled in the direction of the reference frequency.



The Pull-in process

The pull-in range also depends on loop filter.

For passive lag filter:

$\Delta\omega_p = 4/\pi \sqrt{(2\zeta \omega_n K_0 K_d - \omega_n^2)}$

For active PI filter :

$\Delta\omega_p \to$ infinity (because high , theoretically infinite, DC gain)

2.7.4 Pull Out Range $\Delta\omega_{po}$

The pull out range is by definition that frequency step which causes a lock-out if applied to the reference input of the PLL.

An exact calculation of the pull0out range is not possible for the linear PLL. However, simulations on an analog computer have led to an approximation:

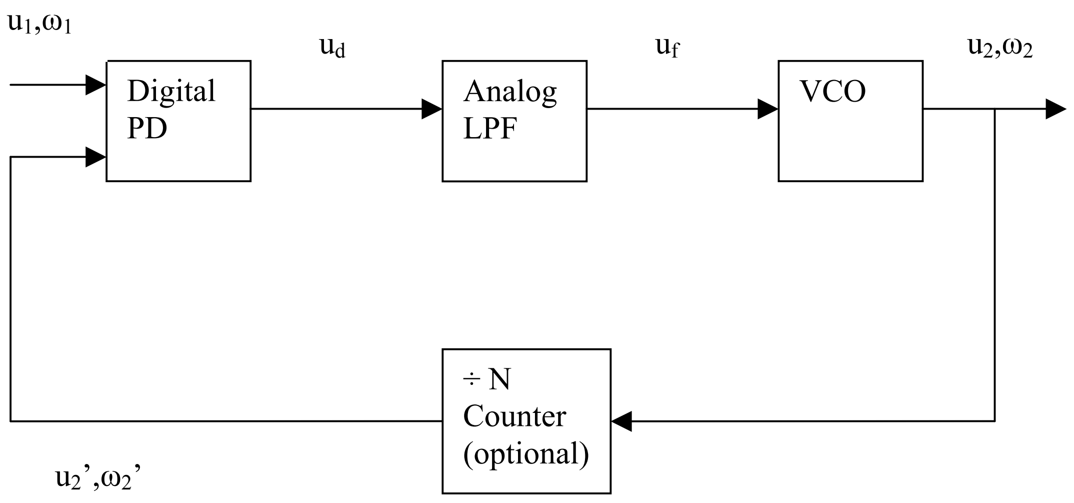
$\Delta\omega_{PO} = 1.8\, \omega_n (\xi + 1)$

In most practical cases the pull out range is between the lock range and the pull in range

$\Delta\omega_L < \Delta\omega_{PO} < \Delta\omega_P$

2.8 THE CALSSICAL DIGITAL PLL (DPLL)

The classical DPLL is actually a hybrid system built from analog and digital functional blocks. The only part of the DPLL that is really digital is phase detector. In many aspects the DPLL performs similar to the LPLL, so some part of PLL theory can be adopted; in some particular aspects, however, DPLL behavior is completely different.



Block diagram of the DPLL

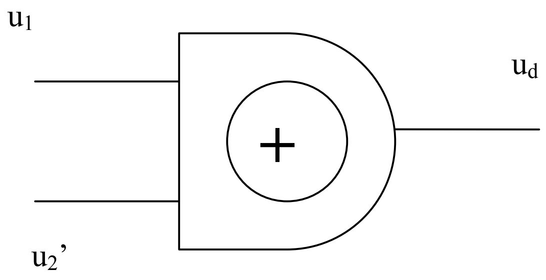
The block diagram of DPLL is shown in figure above, like the LPLL, consists of the three known function blocks phase detector, loop filter and voltage controlled oscillator. In many DPLL applications (e.g.,PLL frequency synthesizers) a divide-by-N counter is inserted between VCO and phase detector. When such a counter is used, the VCO generates a frequency which is N times the reference frequency.

2.8.1 DIGITAL PHASE DETECTORS

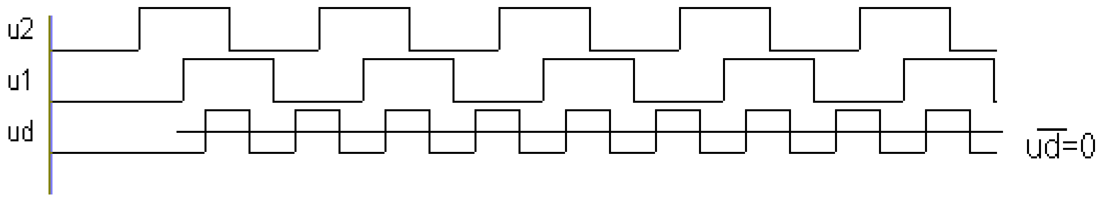
Three most important logical circuits for phase detector are :

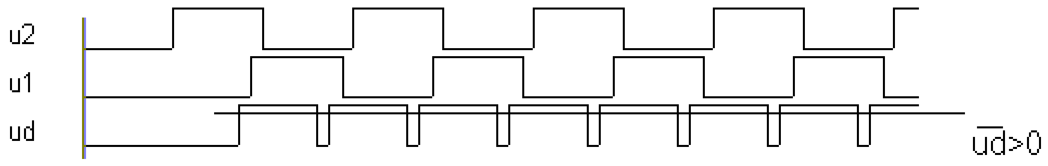
- The EXOR gate
- The (edge triggered) JK flip-flop
- The "phase frequency detector" (PFD)

2.8.1.1 EXOR Phase detector



The operation of EXOR phase detector is most similar to that of the liner multiplier. The signals in DPLLs are always binary signals, i.e., square waves. We assume for the moment that both signals u_1 and u_2 are symmetric square waves.





At zero phase error the signals u_1 and u_2 are out of phase by exactly 90°. Then the output signal u_d is a square wave whose frequency is twice the reference frequency; the duty cycle of the u_d signal is exactly 50%. Because the high frequency component of this signal will be filtered out by loop filter, we consider only the average value of u_d, as shown by dashed line. The average value is arithmatic mean of the two logical levels.

When the output signal u_2' lags the reference signal u_1, the phase error θ_e becomes positive, the duty cycle of u_d becomes larger than 50% i.e., average value of u_d is considered positive. Clearly the mean of u_d reaches its maximum value for a phase error of $\theta_e = 90°$ and its minimum value for $\theta_e = -90°$. Whereas the output signal of the four quadrant multiplier varied with the sine of phase error, the average output of $\bar{u}_d$ of the EXOR is a triangular function of the phase error. Within a phase error range of $-\pi/2 < \theta_e < \pi/2$, u_d is exactly proportional to θ_e and can be written as

$\bar{u}_d = K_d\theta_e$.

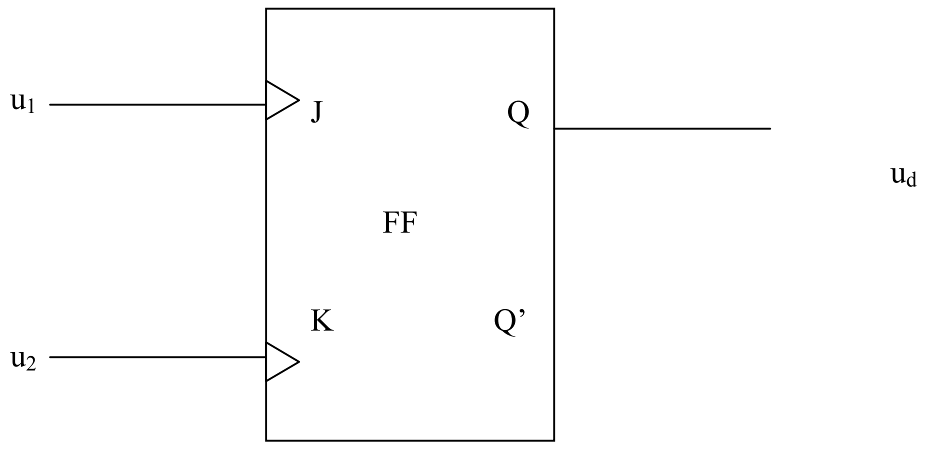
K_d : constant.

When supply voltage to EXOR are U_B and 0, and when we assume logic levels are U_B and 0. K_d is given by

$K_d = U_B/\pi$

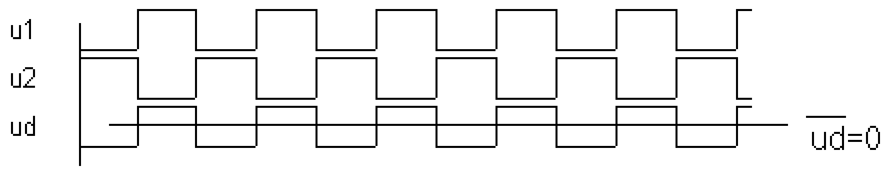
The performance of the EXOR phase detector becomes severely impaired if the signals u_1 and u_2' become asymmetrical. If this happens, the output signal $\bar{u}_d$ gets clipped at some intermediate level, this reduces loop gain of the DPLL and results in smaller lock range and pull out range, etc.

2.8.1.2 J-K Flipflop Phase Detector



JK- Flipflop

Waveform symmetry is unimportant; however the JK-flipflop is used as phase detector. This JK-flipflop differs from conventional JK-flipflops, because it is edge-triggered. A positive edge appearing at the J input triggers the flipflop into its "high" state (Q = 1), a positive edge at the K input into its "low" state (Q = 0).



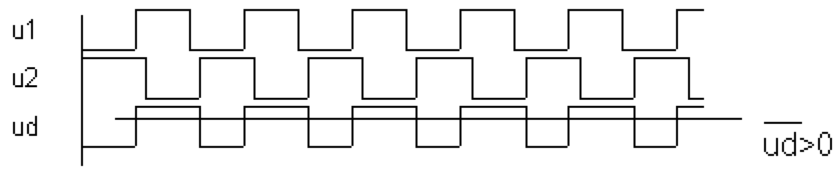


Figure above shows the waveforms of the JK-flipflop phase detector for the case $\theta_e = 0$. With no phase error, u_1 and u_2 have opposite phase. The output signal u_d then represents a symmetrical square wave whose frequency is identical with the reference frequency (and not twice the reference frequency). This condition is considered as $\bar{u}_d$ being zero. If the phase error become positive, the duty cycle of the u_d signal becomes greater than 50 percent, i.e., $\bar{u}_d$ becomes positive. Clearly, $\bar{u}_d$ becomes maximum when phase error reaches 180° and minimum when the phase error is -180°. If the mean value of $\bar{u}_d$ is plotted vs phase error θ_e the saw tooth characteristic is obtained. Within a phase error range of $-\pi < \theta_e < \pi$ the average signal u_d is proportional to θ_e and is given by

$$\bar{u}_d = K_d \, \theta e$$

Obviously the JK-flipflop phase detector is able to maintain phase tracking for phase errors within the range

$$-\pi < \theta_e < \pi$$

By analogous consideration, the phase detector gain of the JK flip flop phase detector is given by $K_d = U_B/2\pi$.

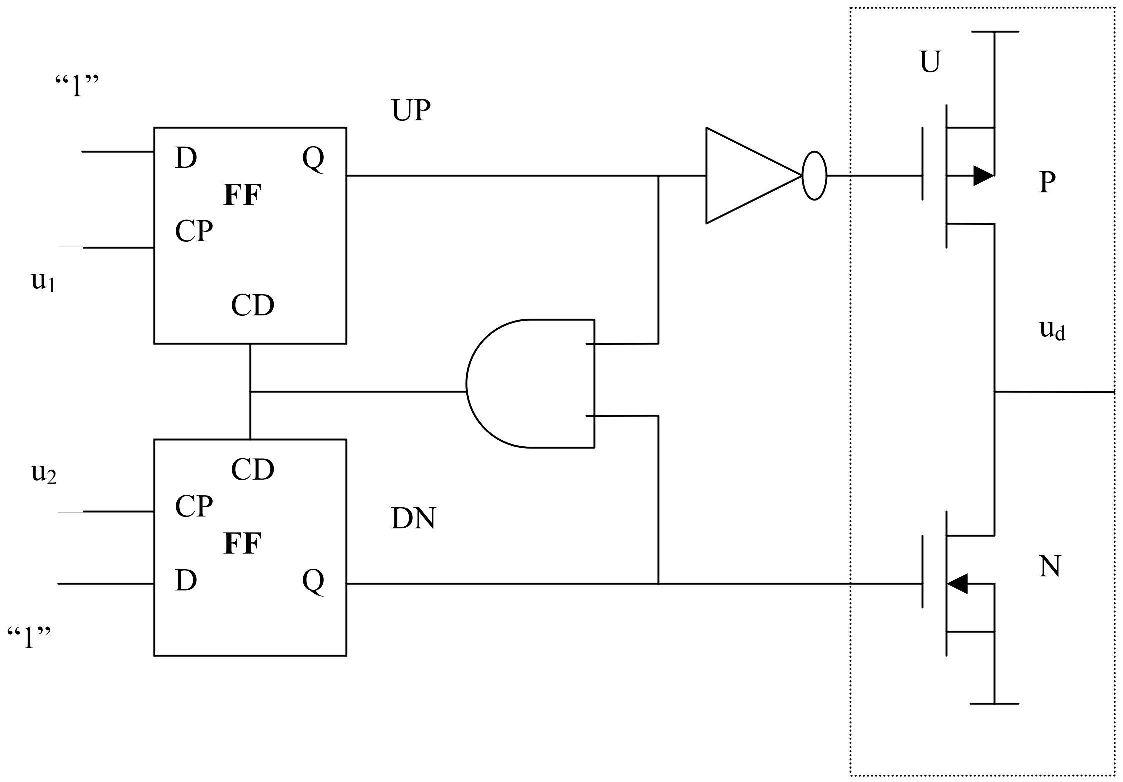
In contrast to EXOR gate , the symmetry of the u_1 and u_2' signals is irrelevant, because the state of the JK flip flop is altered only by the positive transitions of these signals.

2.8.1.3 PFD (Phase Frequency Detector)

As its name implies, its output signal depends not only on phase error θ_e but also on frequency error $\Delta\omega = \omega_1 - \omega_2$', when the DPLL has not yet acquired lock. Fig above shows the schematic diagram of the PFD. It is built from two D-flip flops, whose outputs are denoted “UP” and “DN”(down) respectively. The PFD can be in one of the four states

- UP= 0 , DN= 0 ,
- UP= 1 , DN= 0 ,
- UP= 0 , DN= 1 ,
- UP= 1 , DN= 1

The fourth state is inhibited, however, by an additional AND gate whose output goes to clear direct (CD) pin which resets both flip-flops.

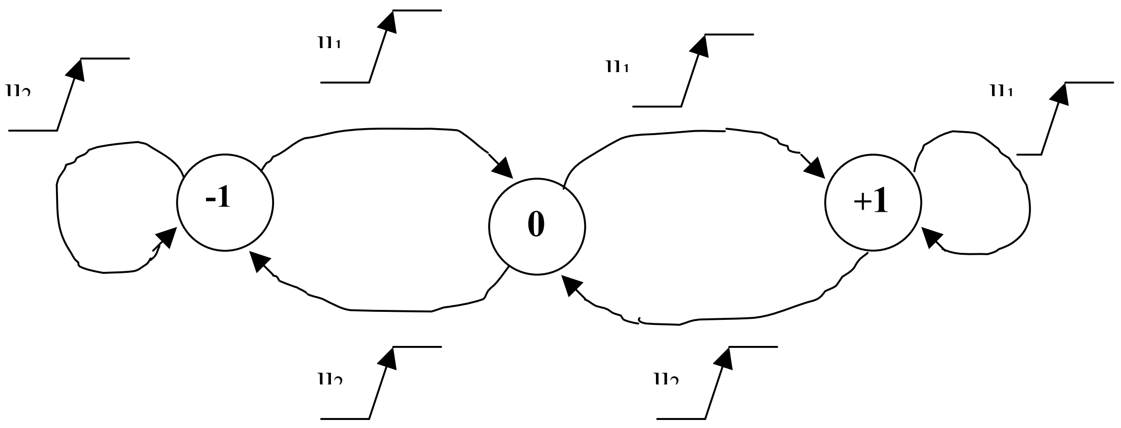


We assign symbols –1, 0, 1 to these states:

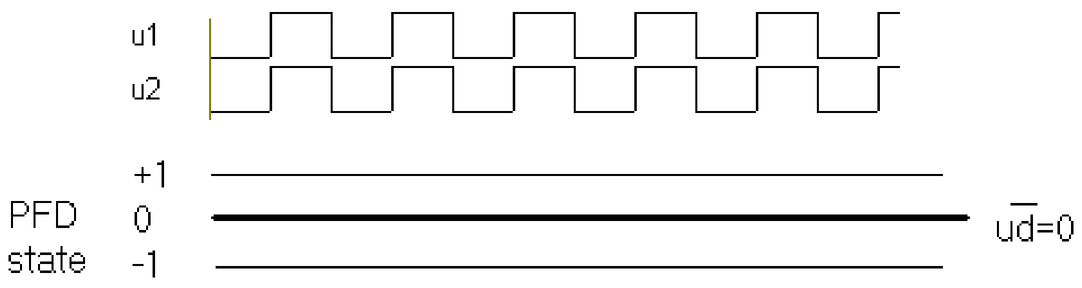
- DN = 1 , UP = 0 → state = -1
- UP = 0 , DN = 0 → state = 0
- UP = 1 , DN = 0 → state = +1

The actual state of PFD is determined by the positive going transients of the signals u_1 and u_2’,as explained by state diagram in fig below. A positive transition of u_1 forces PFD to go into its next higher state unless it is already in +1 state. In analogy, a positive edge of u_2’ forces the PFD into its next lower stae, unless it is already in the –1 state. When PFD is in +1 state, u_d must be positive;

when it is in -1 state, u_d must be negative; and when it is in 0 state, u_d must be zero. Theoritically u_d is a ternary signal. Third state can be substituted by a high impedance state.

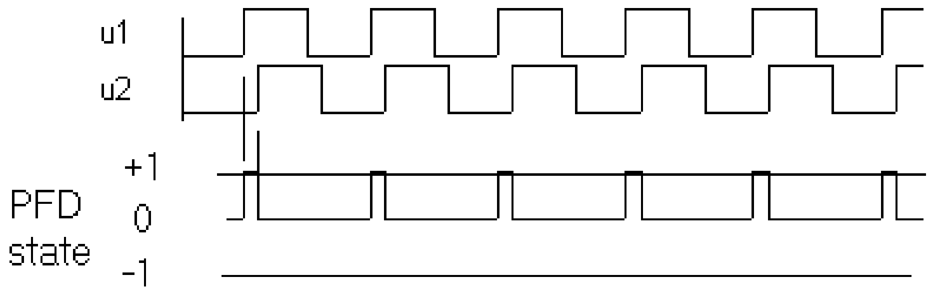


When the UP signal is high, the P channel MOS transistor conducts, so u_d equals the positive supply voltage U_B. When the DN signal is high, the N channel MOS transistor conducts, so u_d is on the ground potential. If neither signal is high, both MOS transistors are off, and the output signal floats i.e., is in the high impedance state. Consequently the output signal u_d represents a tristate signal.
Fig (a) shows the case where the phase error is zero. It is assumed that the PFD has been in 0 state initially. The signals u_1 and u_2' are exactly in phase here; both positive edges of



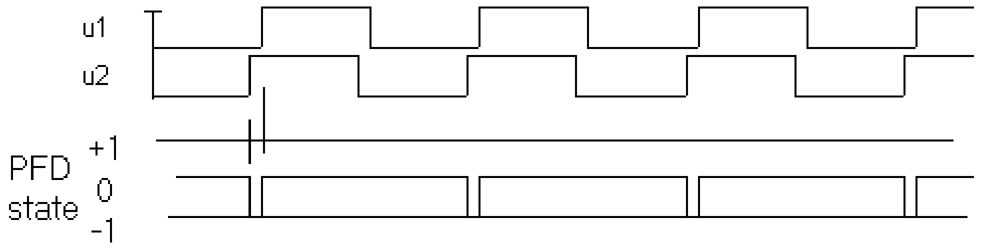
(a)

u_1 and u_2' occur "at the same time"; hence their effects will cancel. The PFD will stay in 0 state for ever. Fig (b) shows the case where u_1 leads u_2'.



(b)

The PFD now toggles between the states 0 and +1. If u_1 lags u_2' as shown in Fig (c),



(c)

the PFD toggles between states –1 and 0. It is easily seen from the waveforms in fig (b) and fig(c) that u_d becomes largest when the phase error is positive and approaches 360° and smallest when the phase error is negative and approaches -360°. When the phase error θ_e exceeds 2π, the PFD behaves as if the phase error is recycled at zero; hence the characteristic curve of PFD becomes periodic with period 2π. An analogous consideration can be made for phase errors smaller then -2π. When the phase error is restricted to the range $-2\pi < \theta_e < 2\pi$, the average of u_d becomes

$\bar{u}_d = K_d\theta_e$

In analogy to JK flip-flop, phase detector gain is computed by

$K_d = U_B/4\pi$

When the logic levels are U_B or 0 respectively.

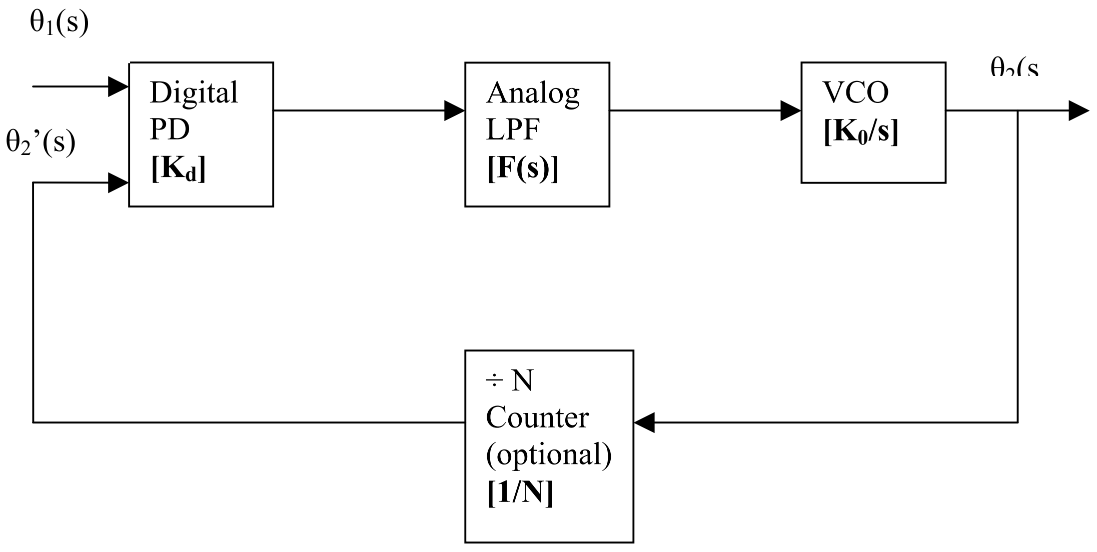
To recognize the bonus offered by the PFD, we must assume that the DPLL is unlocked initially. Furthermore we make the assumption that the reference frequency ω_1 is higher than the output frequency ω_2'.The u_1 signal then generates more positive transitions per unit of time then the signal u_2'. Looking at fig(b) above we see that the PFD can toggle only between the states 0 and +1 under this condition but will never go into the –1 state. If ω_1 is much higher than ω_2' furthermore, the PFD will be in +1 state most of the time. When ω_1 is smaller than ω_2' however, the PFD will toggle between the states –1 and 0. If ω_1 is much lower than ω_2' , the PFD will be in -1 state most of the time. We conclude therefore that the average output signal $\bar{u}_d$ of the PFD varies monotonically with the frequency error $\Delta\omega = \omega_1 - \omega_2'$ when the DPLL is out of lock. This leads to the term phase-frequency detector.

For the case $\omega_1 < \omega_2$ the duty cycle δ is defined the average fraction of time the PFD is in the –1 state ; for $\omega_1 > \omega_2$, δ is by definition minus the average fraction of time the PFD is in the +1 state. As expected, δ approaches –1 when $\omega_1 << \omega_2$ and +1 when $\omega_1 >> \omega_2$.

Because the output signal $\bar{u}_d$ of the PFD depends on phase error in the locked state of the DPLL and on frequency error in the unlocked state, a DPLL which uses the PFD will lock under any condition, irrespective of the type of loop filter used. For this reason the PFD is the preferred phase detectors in DPLLs.

2.8.2 DYNAMIC PERFORMANCE

When the DPLL has acquired lock and is not pulled out by large phase steps, frequency steps, or phase noise applied to its reference input, its performance can be analyzed by a linear model, as done for the LPLL.



Knowing the transfer functions of all building blocks of the DPLL, we are able to drive the phase-transfer function H(s), the natural frequency ω_n and damping factor ξ. For ω_n and ξ , expressions similar to those for the LPLL are obtained.

2.8.3 PARAMETERS

- The Hold Range
- The Lock Range

- Pull in Range
- Pull out Range

2.8.3.1 The Hold Range

The hold range $\Delta\omega_H$ is the frequency range within which PLL operation can be statically stable. Under normal operating conditions the PLL never operates at the limits of the hold range. To reach this limit of stability it would be necessary to weep the reference frequency slowly upward (or downward). If the reference frequency is increased and the dc gain of the loop filter is finite, the phase error increases in proportion. When it attains the maximum value for which the phase detector operates linearly, the hold range is reached.

If an EXOR gate is used as phase detector, the maximum phase error is $\pi/2$. Based on the procedure as was adopted in LPLL case, we obtain for hold range

$$\Delta\omega_H = \frac{K_0 K_d F(0)(\pi/2)}{N}$$

Where F(0) is dc gain of the loop filter.

DC gain is 1 for passive lag,

For EXOR phase detector and passive lag filter, hold range:

$$\Delta\omega_H = \frac{K_0 K_d(\pi/2)}{N}$$

If the JK flip flop is used as phase detector, the maximum phase error becomes π, so hold range :

$$\Delta\omega_H = \frac{K_0 K_d \pi}{N}$$

The situation changes drastically, however, when the PFD is used as phase detector. Because its output is in the high-impedance state when none of the UP or DN outputs is active, the charge on the capacitor(s) of the loop filter remains unchanged when the PFD is in the 0 state. Consequently the output signal u_f of the loop filter can have a non-zero value even if the average u_d signal is 0. When driven by a tri-state source the loop filter acts like an integrator i.e., filter whose transfer function F(s) has a pole at $s = 0$.

The hold range of a DPLL using the PFD becomes infinite.

2.8.3.2 The Lock Range

By definition lock range is the offset between the reference and (scaled down) VCO frequency which causes the DPLL to acquire lock within one beat note between reference and (scaled down) output frequencies. The lock range of the DPLL can be determined by considerations analog to those made in LPLL case we assume DPLL is initially out of lock and that the VCO oscillates on its center frequency $N\omega_0$. The reference frequency is offset by $\Delta\omega$ from its center value ω_0,i.e., $\omega_1 = \omega_0 + \Delta\omega$. The signals u_1 and u_2 can then be replaced by the walsh functions

$u_1 (t) = U_{10}\omega[(\omega_0 + \Delta\omega)t]$

and

$u_2 (t) = U_{20}\omega(\omega_0)t$

respectively, where U_{10} and U_{20} are the amplitudes of the square wave signals. The phase error θ_e is the difference of the phases of these two signals, i.e.,

$\theta_e(t) = \omega_0 t$: which is a ramp function.

Case of EXOR phase detector :

Average output signal $\bar{u}_d$ (t) of the EXOR is a triangular function of phase error, $\bar{u}_d(t)$ becomes triangular function of time. The shape of u_f signal is also triangular, therefore frequency of VCO is modulated by this triangular function. When the frequency offset $\Delta\omega$ is chosen such that the peak of the ω_2' curve just reaches the reference frequency ω_1 , $\Delta\omega$ equals the lock range $\Delta\omega_L$. Using mathematical model developed in LPLL case, we obtain lock range

$\Delta\omega_L \approx \pi\xi\omega_n$

The lock range of the DPLL using the EXOR phase detector is greater than the lock range of the LPLL by a factor of approximately $\pi/2$. This is easily explained by the fact that the maximum output signal of the four quadrant multiplier is K_d, whereas the maximum output signal of the EXOR is $K_d\pi/2$.

Case of JK flip-flop phase detector :

Average output signal $\bar{u}_d(t)$ of the JK flip-flop varies in a saw-tooth-like fashion with phase error, $\bar{u}_d(t)$ will also be a saw-tooth function . The frequency of VCO is modulated in a saw tooth like manner. When the frequency offset $\Delta\omega$ is chosen such that the peak of the ω_2' curve just reaches the reference frequency ω_1 , $\Delta\omega$ equals the lock range $\Delta\omega_L$. By analog consideration we get the approximation

$\Delta\omega_L \approx 2\pi\xi\omega_n$

Case of PFD phase detector :

A similar procedure can be applied to the PFD. For a DPLL using the PFD the lock range becomes approximately

$\Delta\omega_L \approx 2\pi\xi\omega_n$

The lock-in time can be calculated by analog consideration as made for the linear PLL. The lock process is completed within one cycle of the damped oscillation at most, so it is reasonable approximation to state that T_L is one period of the damped oscillation:

$T_L \approx 2\pi/\omega_n$

2.8.3.3 Pull-in range

Pull-in process is a nonlinear phenomenon and is very hard to calculate. Analysis is different for different phase detectors.

Let us assume first EXOR gate phase detector.

We assume DPLL is initially out of lock and that the VCO oscillates on its center frequency $N\omega_0$. The reference frequency is offset by $\Delta\omega$ from its center value ω_0,i.e., $\omega_1 = \omega_0 + \Delta\omega$. The signals u_1 and u_2 can then be replaced by the walsh functions

$u_1(t) = U_{10}\omega[(\omega_0 + \Delta\omega)t]$

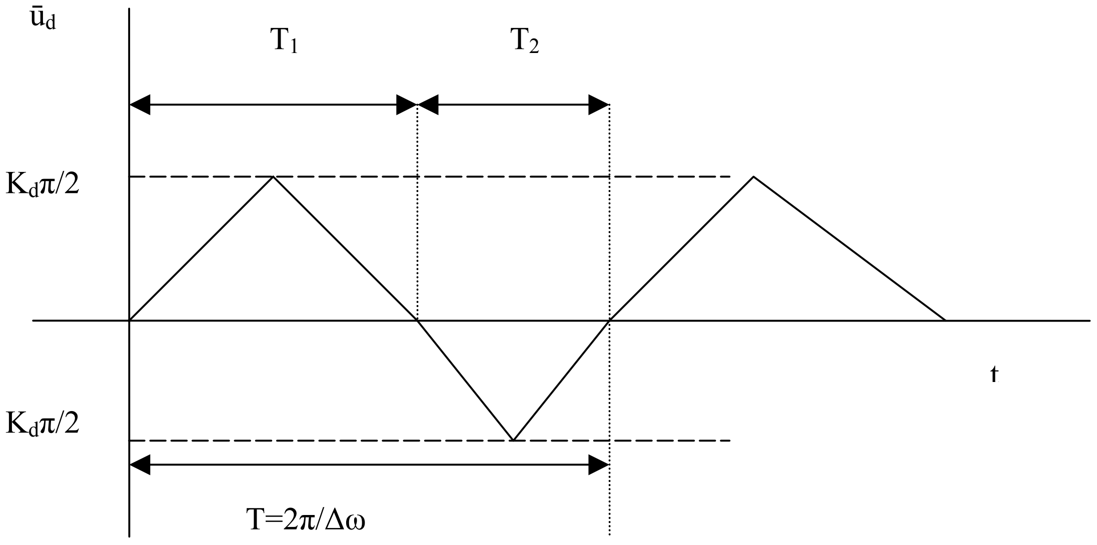
and

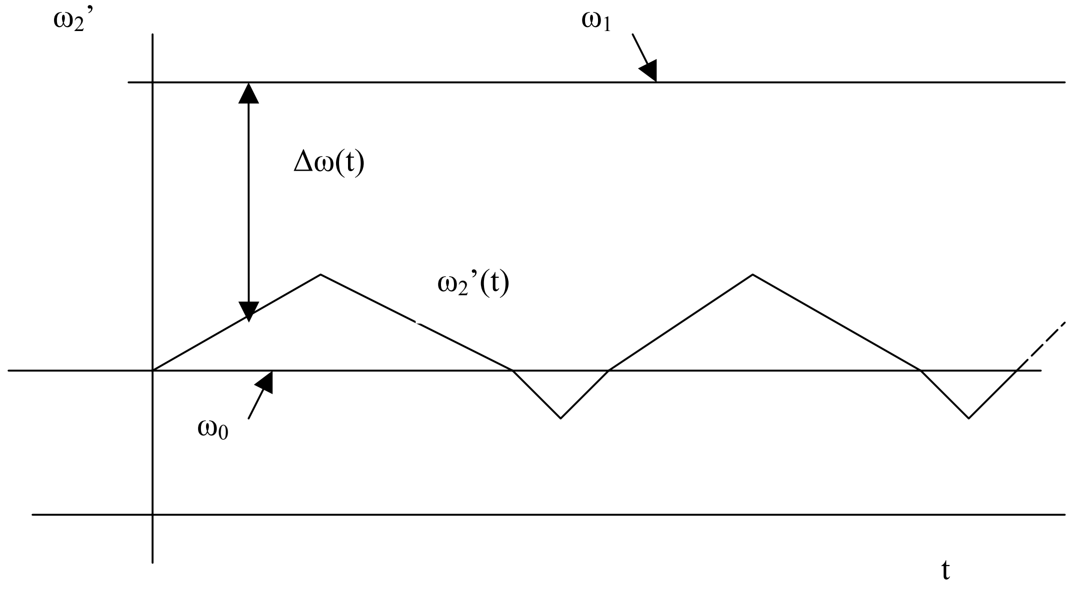
$u_2(t) = U_{20}\omega(\omega_0)t$

respectively, where U_{10} and U_{20} are the amplitudes of the square wave signals. The phase error θ_e is the difference of the phases of these two signals ,i.e.,

$\theta_e(t) = \omega_0 t$: which is a ramp function.

Average output signal $\bar{u}_d(t)$ of the EXOR is a triangular function of phase error, $\bar{u}_d(t)$ becomes triangular function of time. The shape of u_f signal is also triangular, therefore frequency of VCO is modulated by this triangular function. If the triangular waveform were symmetrical the average frequency ω_2' would remain constant and is equal to ω_0 . However the frequency offset $\Delta\omega$ is not constant but is given by the difference between reference frequency ω_1 and instantaneous frequency ω_2'. Consequently, $\Delta\omega(t)$ becomes smaller during the positive half of the $\bar{u}_d$ signal and larger during the negative half-wave. Therefore, the waveform of $\bar{u}_d$ becomes asymmetrical. When the $\bar{u}_d$ waveform is asymmetrical, its mean value is no longer zero but becomes slightly positive. This causes the average frequency of the VCO to be pulled up. If the loop gain is high, pull in process becomes regenerative and the VCO frequency will be pulled up until it becomes close to the reference frequency. Then a locking process will take place. A pull-in process is initiated whenever the initial frequency offset $\Delta\omega$ is smaller than the pull-in range $\Delta\omega_p$.





Final results:

For passive lag low gain loops

$\Delta\omega_p = (\pi/2)\sqrt{(2\xi\omega_n K_0 K_d - \omega_n^2)}$

For passive lag high gain loops

$\Delta\omega_p = (\pi/\sqrt{2})\sqrt{(\xi\omega_n K_0 K_d)}$

Final result for approximate pull-in time T_P :

$T_P = 4\Delta\omega_0^2/(\pi^2\xi\omega_n^3)$

Pull in time varies with square of initial frequency offset. Pull in time becomes infinite when initial frequency offset equals the pull-in range.

In case of JK flip-flop phase detector, the waveforms of average $\bar{u}_d(t)$ signal is saw tooth instead of triangular. Performing analog computation as above, we get for the pull-in range :

For passive lag low gain loops

$\Delta\omega_p = \pi\sqrt{(2\xi\omega_n K_0 K_d - \omega_n^2)}$

For passive lag high gain loops

$\Delta\omega_p = (\pi/\sqrt{2})\sqrt{(\xi\omega_n K_0 K_d)}$

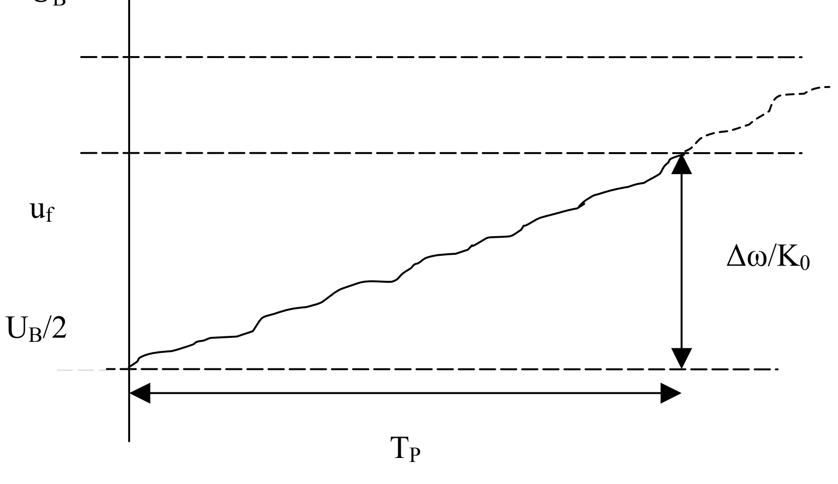
Pull in time

$T_P = \Delta\omega_0^2/(\pi^2\xi\omega_n^3)$

Consider the case where PFD is used as phase detector:

The pull-in range becomes infinite now, because the loop filter is driven by a tri-state source. The charge on the filter capacitor remains unchanged when the output of the PFD is in the high impedance state, Hence even a passive lag filter works like a real integrator.

We assume again that the DPLL is initially out of lock and that the frequency ω_1 of the reference signal u_1 is marked higher than the (down scaled) output frequency ω_2'. The output signal u_d of the PFD then toggles between the states 0 and +1.The average u_d signal has the shape of saw-tooth signal. It periodically ramps up from 0 to 1 and is a saw-tooth function as well. The average duty cycle of u_d is 50%. Because the time constant τ_1 of the loop filter is much larger than the period of the u_1 signal, an equivalent u_{eq} having a constant duty cycle of 50% would have the same effect on the loop filter (neglecting τ_2 because $\tau_2 << \tau_1$). Because the duty cycle is only 50%, however the capacitor needs twice as much time to charge. Therefore the loop filter acts like a simple RC filter whose time constant is not τ_1 but $2\tau_1$. After some time, u_f will have reached a level which causes the VCO to generate just the "right" frequency.



This occurs when the voltage on the capacitor has been increased by the amount $\Delta\omega/K_0$. When this happens, the pull-in process terminates, and a lock-in process takes place.

For passive lag loop filter pull-in time (time required for the capacitor to increase its voltage by $\Delta\omega/K_0$) :

$T_P = 2\tau_1 \ln((K_0U_B/2)/K_0U_B/2 - \Delta\omega_p)$

Where $\Delta\omega_p$ is the initial frequency offset, $\Delta\omega_p = \omega_1 - \omega_p$

Major difference of the pull-in process for different types of phase detectors :

If the phase detector is an EXOR gate, the instantaneous frequency of the VCO is modulated in both directions around its average value. Provided a pull-in process starts, the frequency of the VCO is slowly "pumped up" as in case of LPLL. A similar pumping is observed when the phase detector is a JK flip-flop. No pumping occurs when the PFD is used, the VCO "knows where to go" at every time. The instantaneous frequency of the VCO approaches the final value from one side only. When the pull-in process is completed, a lock-in process follows.

2.8.3.4 Pull-out range

Pull out range is the size of the frequency step applied to the reference input which causes the PLL to loose phase tracking.

In case of LPLL the output signal of four quadrant multiplier varies with sine of the phase error, it is triangular function in case of EXOR which is quite similar. We expect, however, that the pull-out range would be slightly greater for the DPLL, because the output signal of the EXOR is linear over the full range $-\pi/2 < \theta_e < \pi/2$ and does not flatten out at phase errors approaching $\pi/2$.

By simulations, using damping factors in the range of $0.1 < \xi < 3$, then a least squares fit gave the approximation

$\Delta\omega_{PO} = 2.46\ \omega_n(\xi + 0.65)$

In case of the JK flip-flop, the pull-out range is the frequency step causing the peak phase error to exceed π ; in case of PFD, the pull-out range is the frequency step leading to a peak phase error of 2π.

Least square fit gives :

For JK flip flop phase detector

$\Delta\omega_{PO} = 5.78\ \omega_n(\xi + 0.5)$ and

For PFD phase detector

$\Delta\omega_{PO} = 11.55\ \omega_n(\xi + 0.5)$

2.8.4 DPLL DESIGN

- Step 1. The input and output frequencies of the DPLL must be specified.
- Step 2. The scalar ratio must be determined.
- Step 3. Determination of damping factor ξ.(0.7 preferred for butterworth response)
- Step 4. Choice of Phase Detector (PFD preferred)
- Step 5. Characteristic of VCO is determined. ω_0 and N are decided, range for ω is generated, VCO gain K_0 is determined
- Step 6. Specify type of loop filter.(passive lag filter is preferred with PFD for infinite hold and pull-in range)
- Step 7. Determining dynamic properties of DPLL. How the DPLL is used, T_P should be key parameter (goto step 8) or $\Delta\omega_{po}$ should be key parameter (goto step 12) or T_L should be key parameter (goto step 13). User must resort to specification which makes as much sense as possible.
- Step 8. With T_P known τ_1 is calculated.
- Step 9. With τ_1 known ξ is calculated.
- Step 10. With ω_n and ξ known τ_2 is calculated.
- Step 11. With τ_1 and τ_2 (plus eventually K_0), the components of loop filter can be determined.
- Step 12. Given $\Delta\omega_{po}$ and damping factor ξ , natural frequency ω_n is calculated. Proceed to step 14.
- Step 13. ω_n is calculated from T_L, proceed to step 14
- Step 14. With ω_n and ξ known τ_1 is calculated.

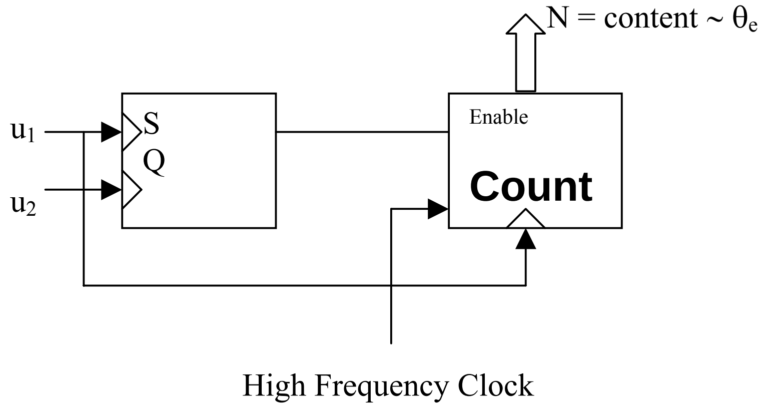
2.9 ALL DIGITAL PLL (ADPLL)

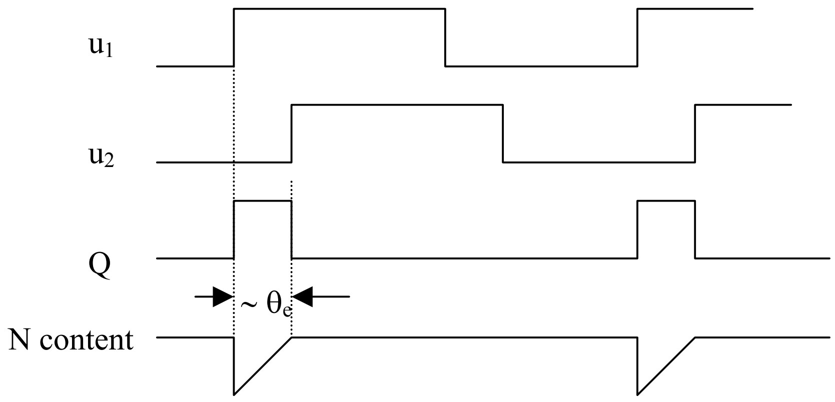
The Classical DPLL is a semi analog circuit. Because it always needs a couple of external components, its key parameters will vary because of parts spread. Even worse, the center frequency of a DPLL is influenced by parasitic capacitors on the DPLL chip. Its variation can be so large that trimming can become necessary in critical applications. Many parameters are also subject to temperature drift.

The all-digital PLL does away with these analog-circuitry headaches. In contrast to the DPLL, it is an entirely *digital* system. Let us know first that the term “digital” is used here for a number of different things. First of all, “digital” means that the system consists exclusively of logical devices. But “digital” also signifies that the signals within the system are digital too. Hence it can be a binary signal (or “bit” signal) as was the case with the classical DPLL, but it can as well be a “word” signal, i.e., a digital code word coming from a data register, from the parallel outputs of a computer, and the like. When discussing the various types of ADPLL, we find the whole palette of such digital signals. To realize an ADPLL, all function blocks of the system must be implemented by purely digital circuits.

2.9.1 DIGITAL PHASE DETECTORS

2.9.1.1 Flip-Flop counter phase detector





The reference (input) signal u_1 and the output (or scaled down output) signal u_2 of the DCO (or VCO) are binary valued signals. They are used to set or rest an edge triggered RS flip flop. The time period in which the Q output of the flip flop is a logic 1 is proportional to the phase error θ_e. The Q signal is used to gate the high frequency clock signal into the upward counter. Note that the counter is reset on every positive edge of the u_1 signal.

The content N of counter is also proportional to the phase error θ_e, where N is n-bit output of this type of phase detector. The frequency of the high frequency clock is usually Mf_0 where f_0 is the frequency of reference signal and M is large positive integer.

2.9.2 DIGITAL LOOP FILTERS

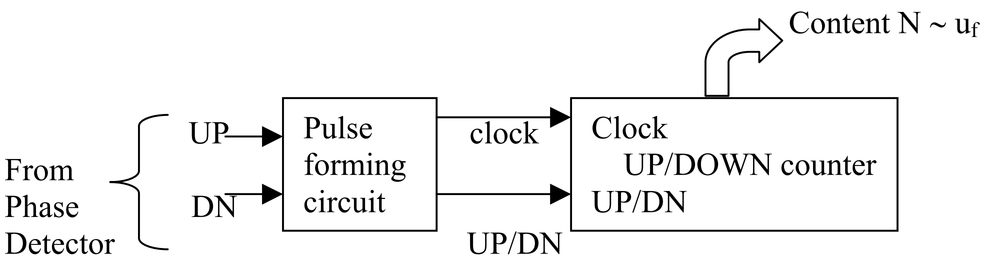
2.9.2.1 UP/DOWN counter filter

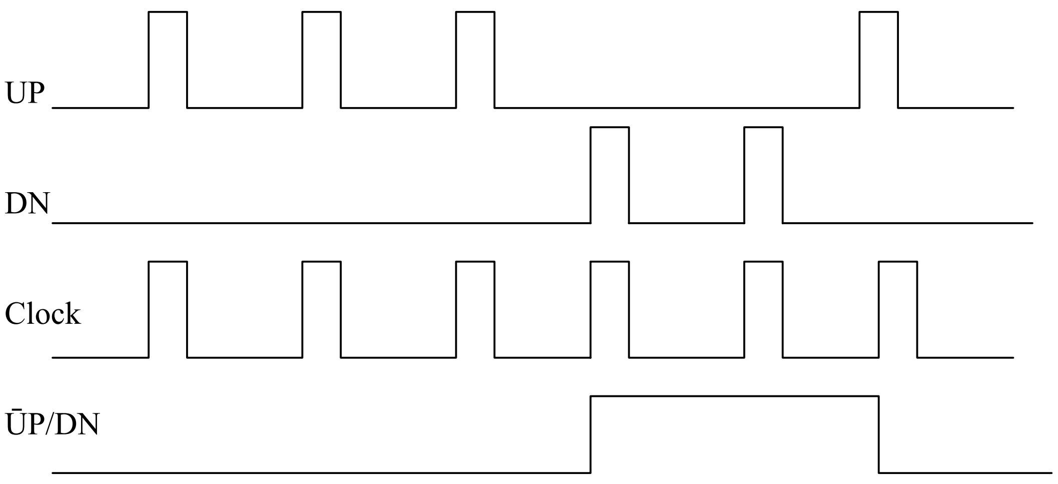
Probably the simplest loop filter is built from an ordinary UP/DOWN counter. The UP/DOWN counter loop filter preferably operates in combination with a phase detector delivering UP or DN (DOWN) pulses, such as the PFD. It is easily adapted, however, to operate in conjunction with the XOR or JK flip flop phase detectors and others, as shown in figure. A pulse-forming network is first needed which converts the incoming UP and DN pulses into a counting clock and a direction ($\overline{\text{U}}$P/DN) signal as explained by waveforms in figure.

On each UP pulse generated by the phase detector, the content N of UP/DOWN counter is incremented by 1. A DOWN pulse will decrement N in same manner. The content N is given by the n-bit parallel output signal u_f of the loop filter. Because the content N is the weighted sum of the UP and DN pulses — the UP pulses have an assigned weight of +1 , the DN pulses, -1 — this filter can roughly be considered an integrator having the transfer function

$H(s) = 1/ sT_I$

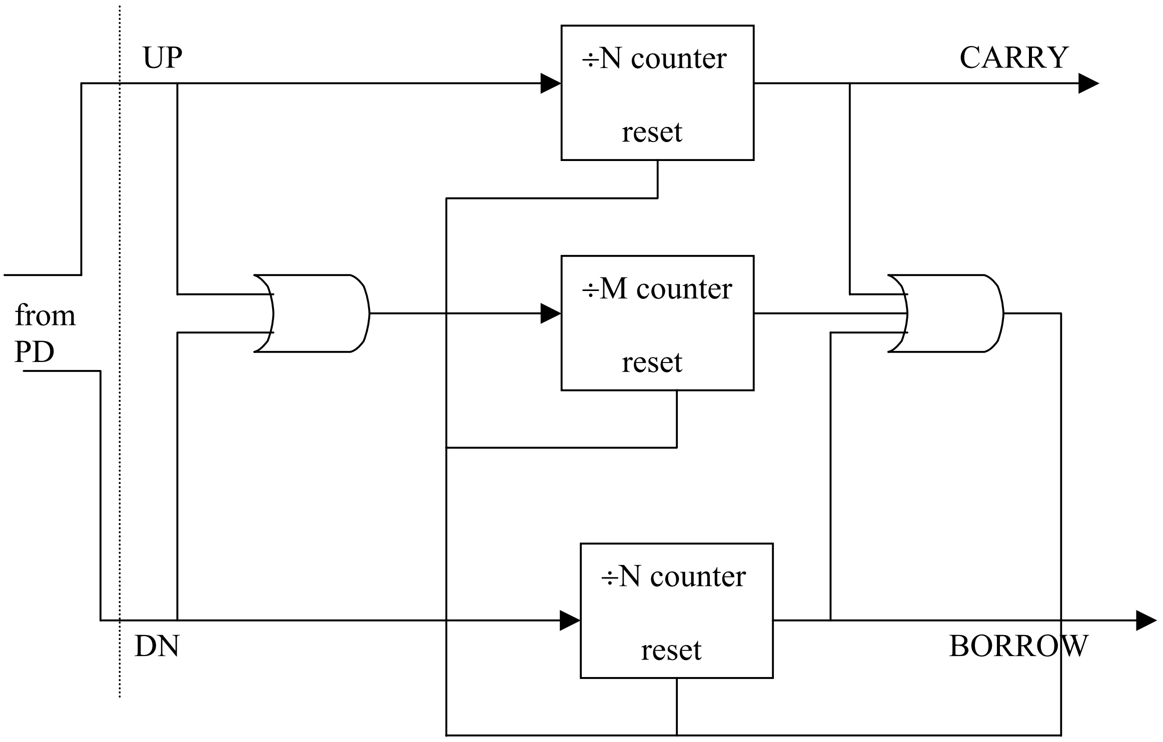
where u_I is the integrator time constant. This is however, a very crude approximation, since the UP and DN pulses do not carry any information about the actual size of the phase error; they only tell whether the phase of u_1 is leading or lagging u_2.





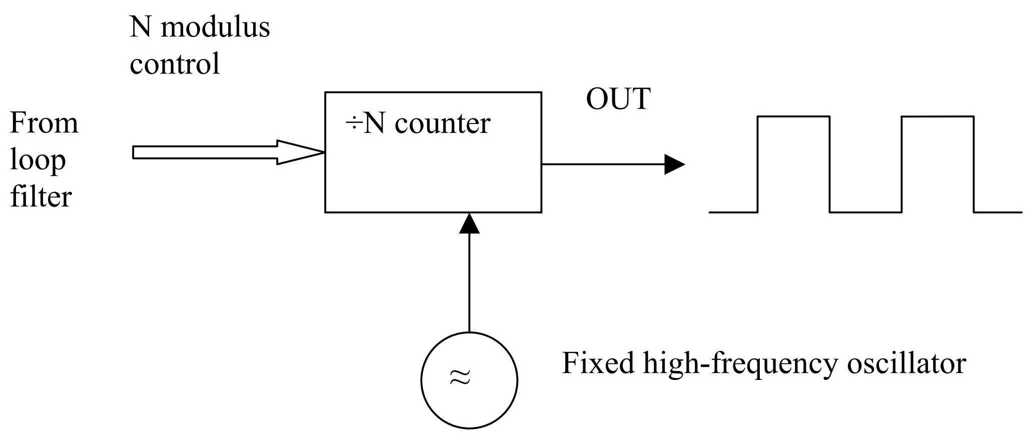
2.9.2.2 N-before-M counter filter

Another digital loop filter is the so called N-before-M counter (shown in figure below). The performance of this filter is very non-linear.It is suggested that N-before-M filter operates in conjunction with a phase detector generating UP and DOWN pulses, as was the case with the PFD. The N-before-M filter uses two frequency counters scaling down the input signal by a factor N and one counter scaling down by M, where M > N always. The ÷M counter counts the incoming UP and DN pulses. The upper ÷N counter will produce one carry output when it has received N UP pulses. But it will generate this CARRY only when the ÷M counter does not receive M pulses. Otherwise the ÷N counter would have been reset. We can say the upper ÷N counter will produce a carry pulse whenever more than N pulses of an ensemble of M pulses have been UP pulses. A similar statement can be made for the lower ÷N counter in figure, which will output BORROW pulses only when the majority of incoming pulses are DN pulses.
The outputs of the N-before-M filter can be used in a similar way to control a DCO, as indicated for the K counter.



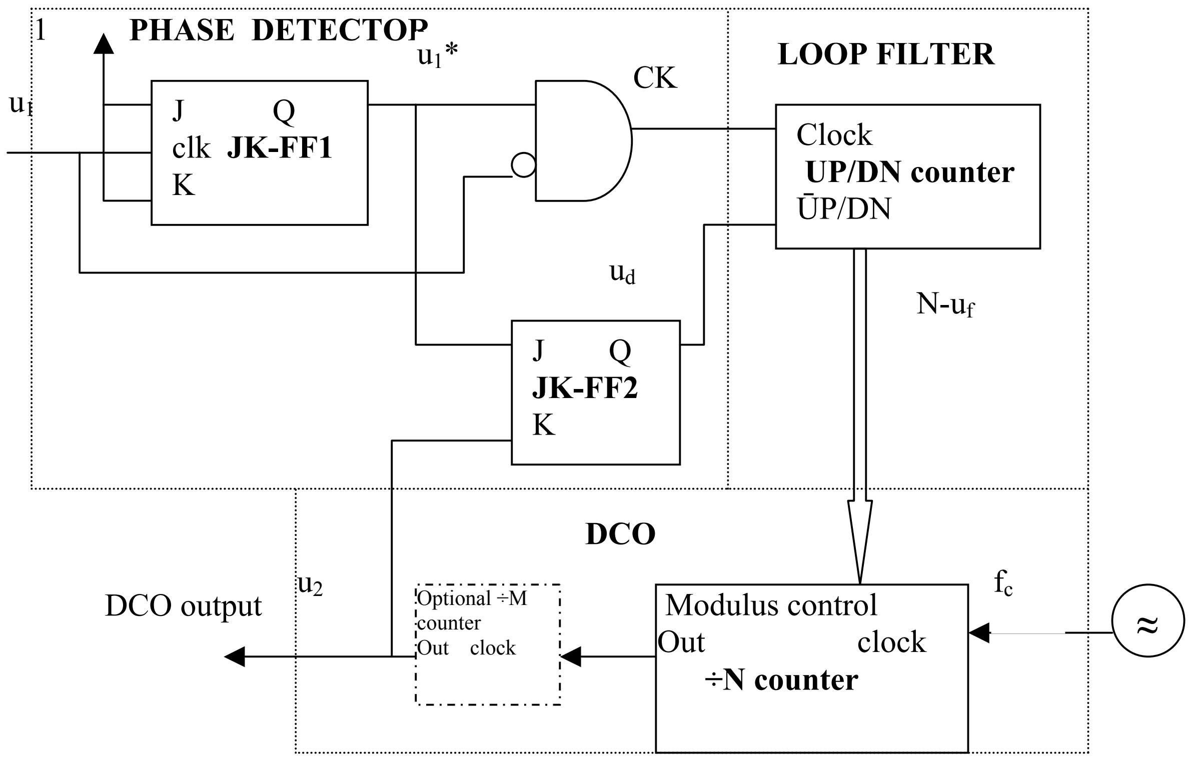
2.9.3 DIGITAL CONTROLLED OSCILLATORS

A variety of DCO’s can be designed; they can be implemented by hardware or software.
Probably the simplest solution is the ÷N counter DCO. A ÷N counter is used to scale down the signal generated by a high frequency oscillator operating at a fixed frequency. The N-bit parallel output signal of a digital loop filter is used to control the scaling factor N of the ÷N counter.



2.9.4 EXAMPLE OF AN IMPLEMENTED ADPLL

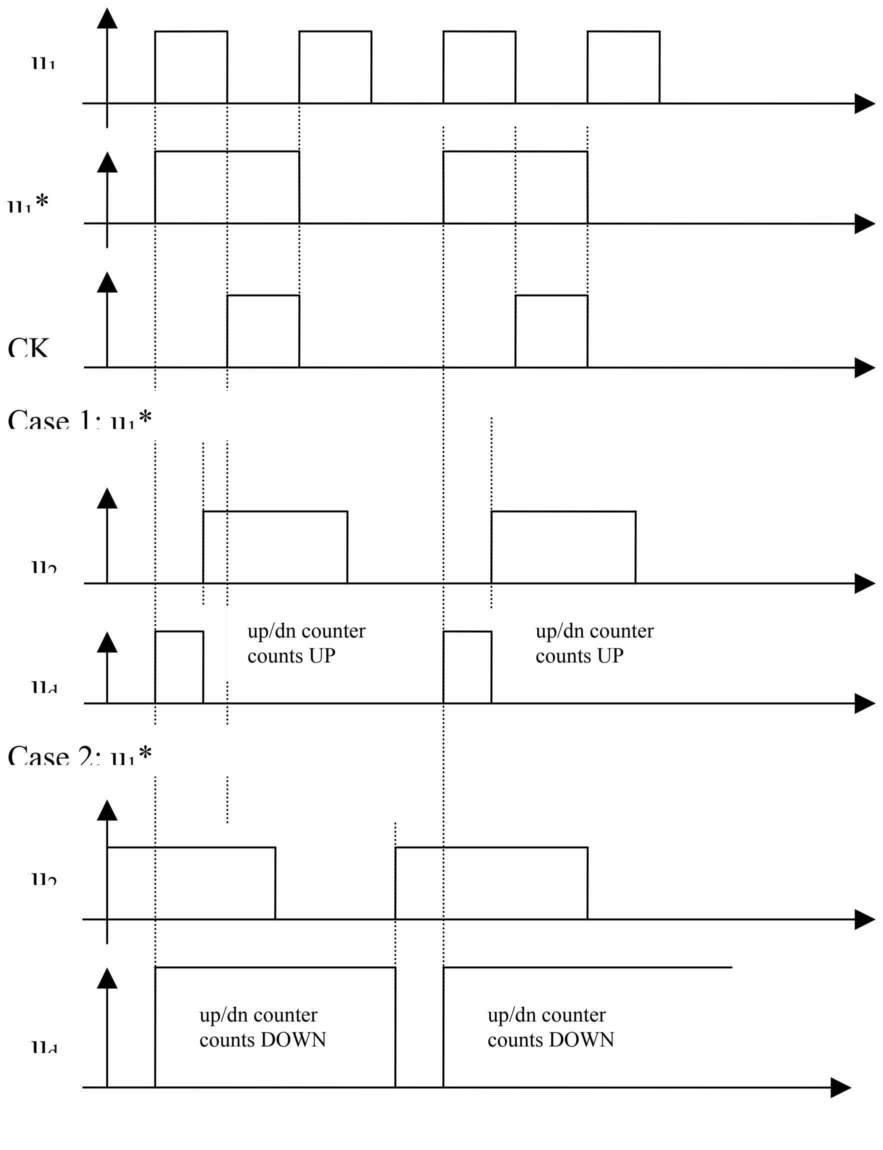
The Example depicted in Figure below, a JK flip-flop (JK-FF2) is used as a phase detector. The Phase detector block has been extended, however, by a pulse forming network consisting of another JK flip flop (JK-FF1) and an AND gate. The loop filter is built from an UP/DOWN counter. ÷N counter is used as DCO. The pulse forming network generates the counting clock (CK) for the UP/DOWN counter.



(a) Block Diagram

The waveforms in figure (b) explain the operating principle of the PD. The JK-FF$_1$ scales down the input signal u_1 by a factor of 2; the scaled down input signal is designated u_1*.The reconstructed signal u_2 has the same frequency as u_1* not u_1. The waveforms in have been drawn for two cases:

1. u_1* leading u_2
2. u_1* lagging u_2



(b) Waveforms

The JK-FF2 is an edge-triggered flip-flop.The positive transitions of u_1* set this flip flop; the positive transition of u_2 reset it. The counting clock CK for the UP/DOWN counter occurs at a time when the output signal u_d or JK-FF2 is stable, that is, high when u_1* lags u_2 or low when u_1* leads u_2. Consequently the phase-detector output signal u_d is used as the direction input $\bar{\text{U}}$P/DOWN for the UP/DOWN counter. If the frequencies of u_1* and u_2 are not identical, the UP/DOWN counter will count upward or downward until N has reached the value that causes ÷N counter to generate the correct output frequency.

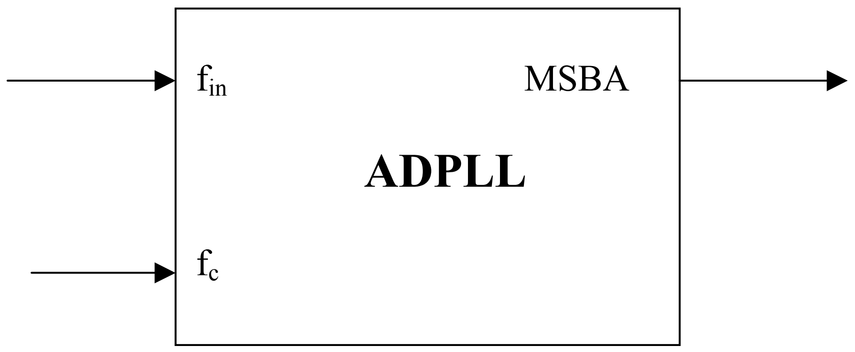
Because N can only be varied in steps of 1, the frequency of signal u_2 will normally be slightly too high or too low. This will force the contents of N to jitter continuously around the values of N and N+1 if the reference frequency f_1 is constant. At equilibrium f_1 will be equal to $f_c/(N-M)$, where M is the scaling factor of the optional ÷M counter.

3. ADPLL IMPLEMENTATION

ADPLL implemented by VHDL ,synthesized and tested on Xilinx FPGA.

3.1 BLOCK DIAGRAM

ADPLL :



This is the symbol of ADPLL synthesized on xilinx FPGA and tested on PCB.

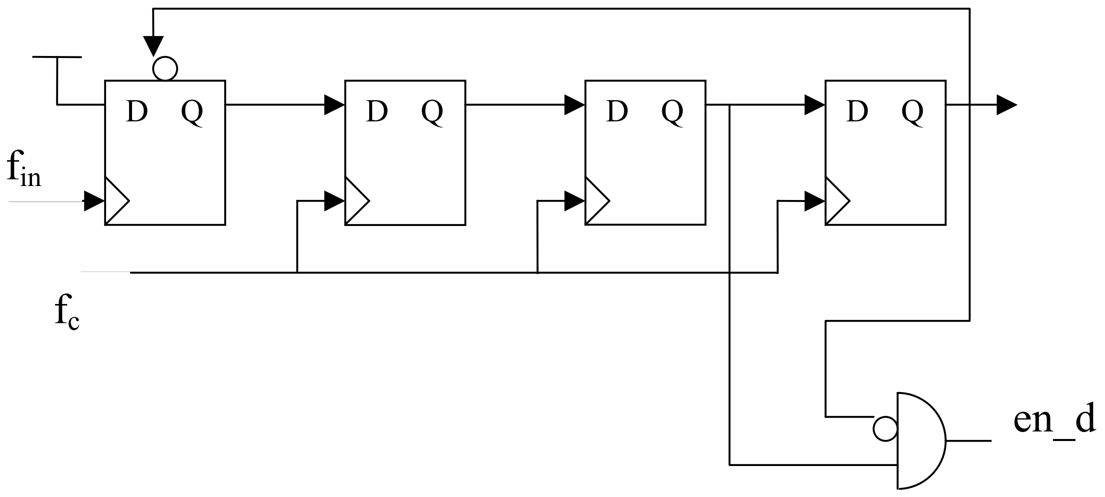
f_{in} is the input reference signal and MSBA is output clock signal locked to input.

f_c is standard high frequency clock from any independent clock source.

The ADPLL comprises of following components:

- Sampling Phase detector
- Loop filter and DCO control unit
- DCO circuit

3.2 SAMPLING PHASE DETECTOR



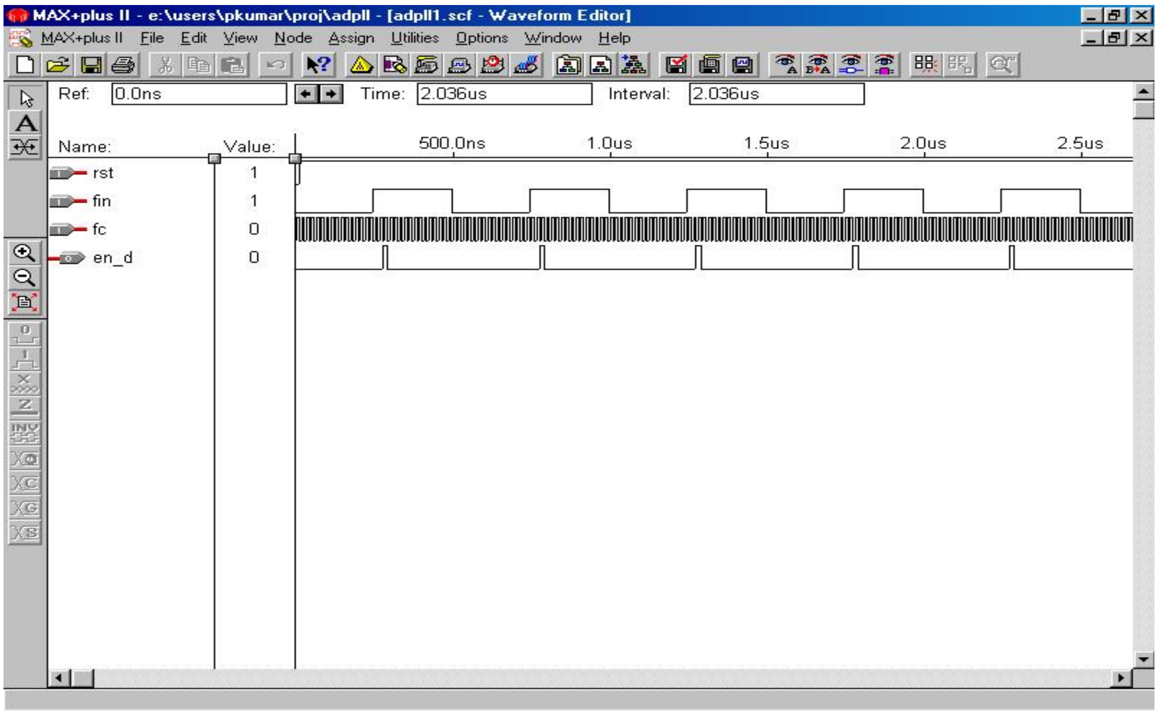
This is the logical diagram of ‘Sampling Phase Detector’ . In this case input reference f_{in}
is sampled by a high frequency (32 or 64 times) f_c. It produces a pulse whenever a positve edge of input reference is sensed.

The circuit was implemented by VHDL and simulated on Altera’s MaxPlus-II tool for various conditions i.e., when f_c is leading f_{in} or when f_c is lagging f_{in} or when f_c is in phase with f_{in} etc.

Following is the VHDL code and simulation results for various cases. An output pulse is generated on every positive edge of f_{in}.

```
library IEEE;
use IEEE.std_logic_1164.all;
-- use IEEE.std_logic_misc.all ;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity adpll is
port(
      fc        : in std_logic;
      fin       : in std_logic;
      rst       : in std_logic;
      en_d      : out std_logic
   );
end adpll;

architecture RTL of adpll is

signal f1out        : std_logic;
signal f2out        : std_logic;
signal f3out        : std_logic;
signal en           : std_logic;
begin
process(en, rst, fin)
begin
        if (rst='0' or en='1') then
               f1out <= '0';
         elsif fin'event and fin = '1' then
           f1out <= '1';
        end if;
end process;

process(f1out, f2out, f3out, fc, rst)
begin
        if rst = '0' then
               f2out <= '0'; f3out <= '0'; en <= '0' ;
         elsif fc'event and fc = '1' then
           f2out <= f1out ; f3out <= f2out ; en <= f3out ;
        end if;
end process;
en_d <= f3out and (not en);
end RTL;
```

PHASE DETECTOR SIMULATIONS :

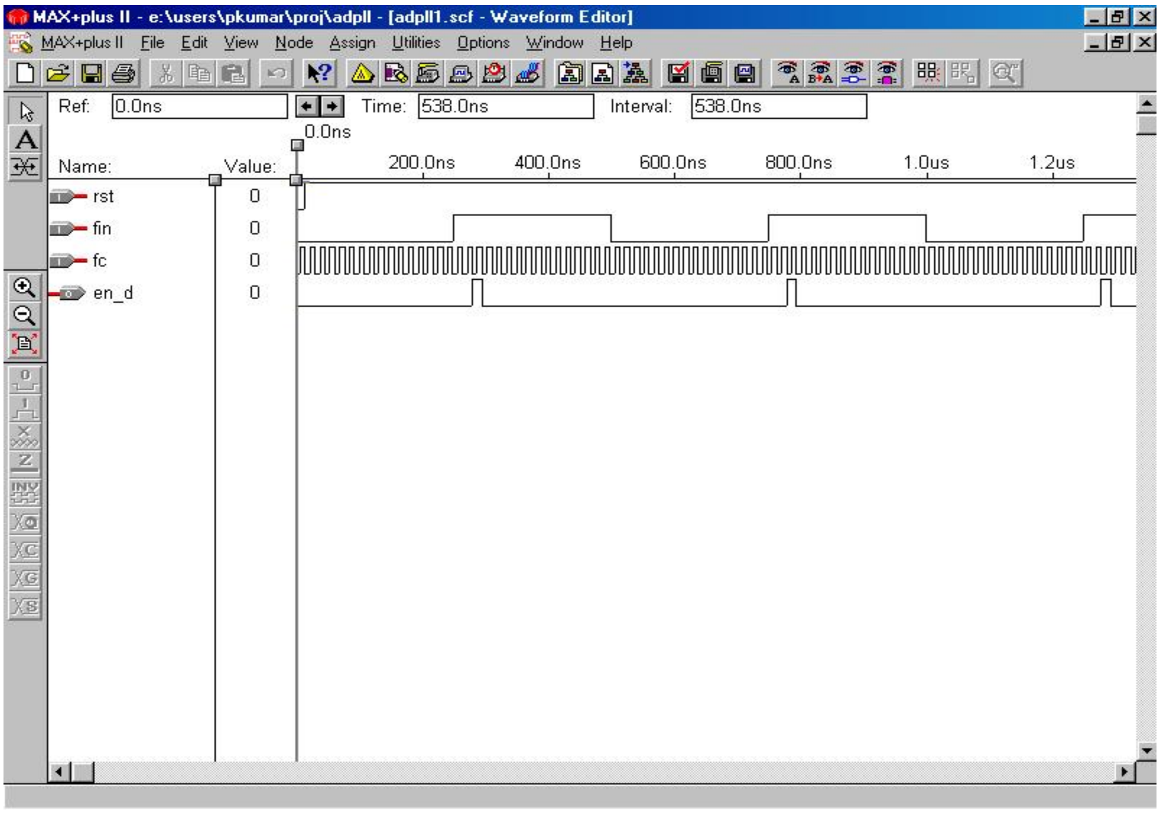


MAX+plus II - e:\users\pkumar\proj\adpll - [adpll1.scf - Waveform Editor]

Ref: 0.0ns Time: 2.036us Interval: 2.036us

```
Name:     Value:
rst       1
fin       1
fc        0
en_d      0
```

MAX+plus II File Edit View Node Assign Utilities Options Window Help
500.0ns
1.0us
1.5us
2.0us
2.5us

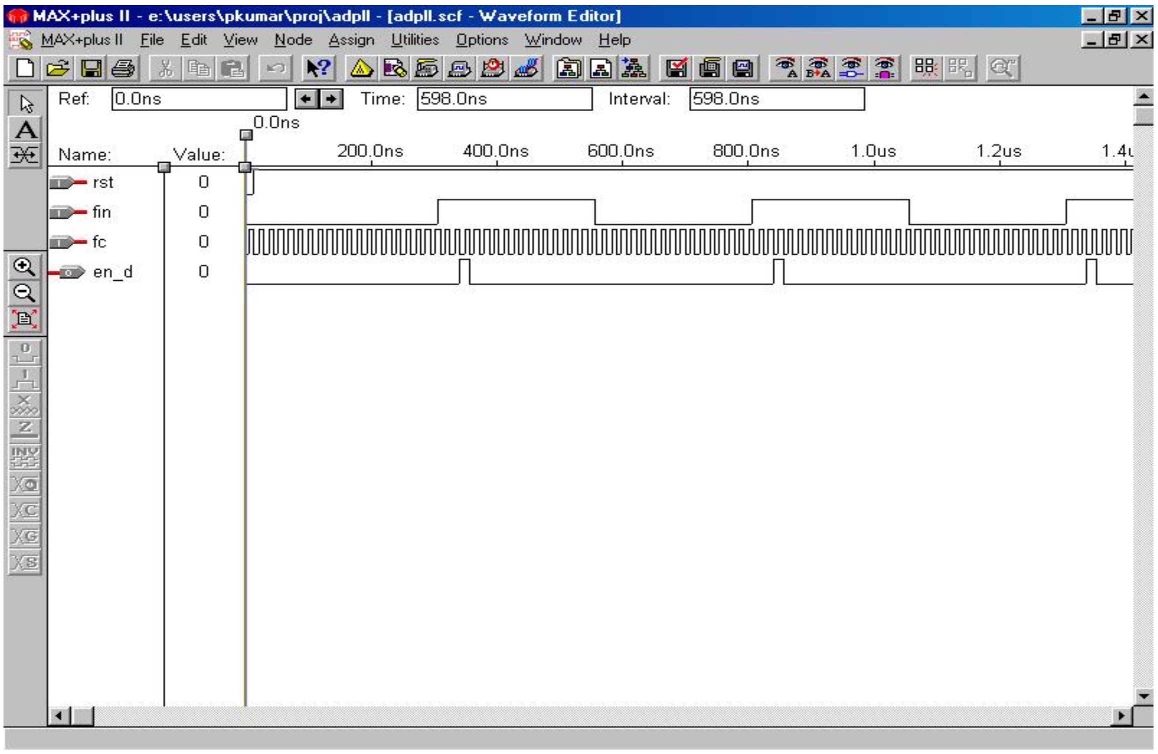


MAX+plus II - e:\users\pkumar\proj\adpll - [adpll1.scf - Waveform Editor]

Ref: 0.0ns Time: 538.0ns Interval: 538.0ns

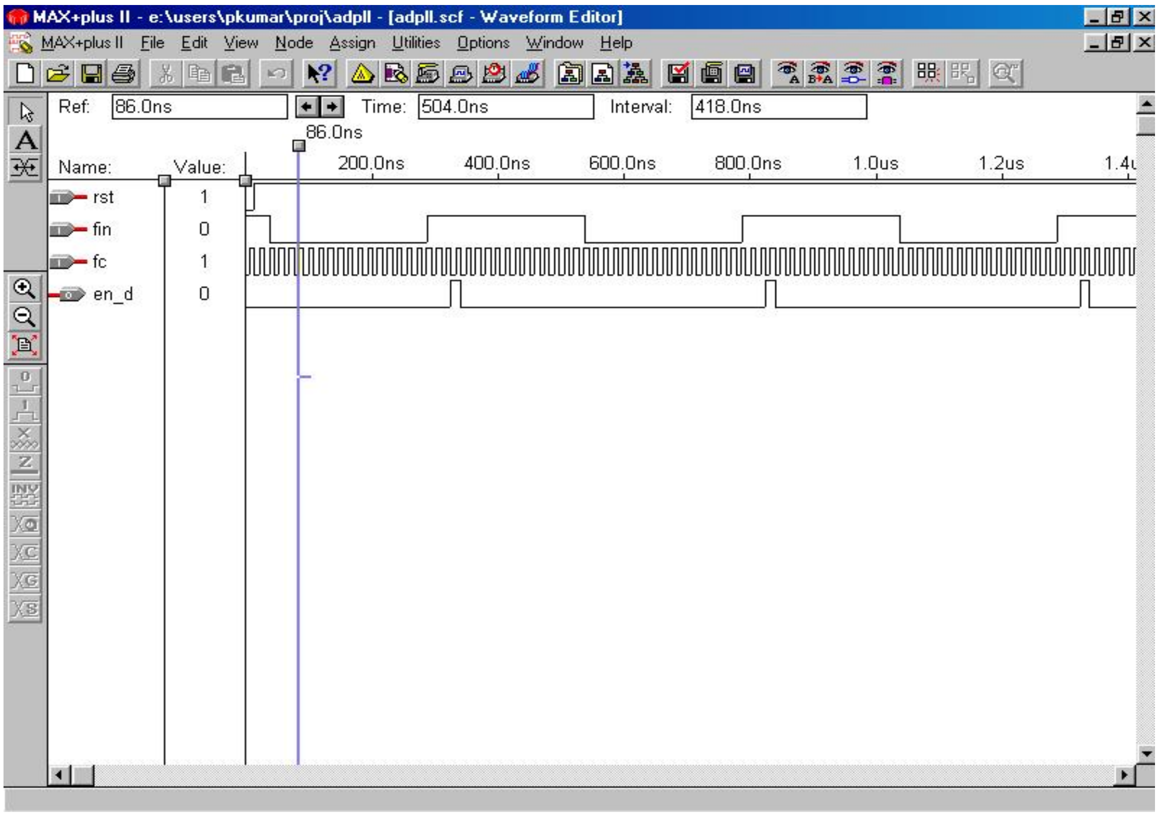
```
Name:     Value:
rst       0
fin       0
fc        0
en_d      0
```

MAX+plus II File Edit View Node Assign Utilities Options Window Help
0.0ns
200.0ns
400.0ns
600.0ns
800.0ns
1.0us
1.2us


MAX+plus II - e:\users\pkumar\proj\adpll - [adpll.scf - Waveform Editor]
MAX+plus II File Edit View Node Assign Utilities Options Window Help
0.0ns
200.0ns
400.0ns
600.0ns
800.0ns
1.0us
1.2us
1.4u

Ref: 0.0ns Time: 598.0ns Interval: 598.0ns

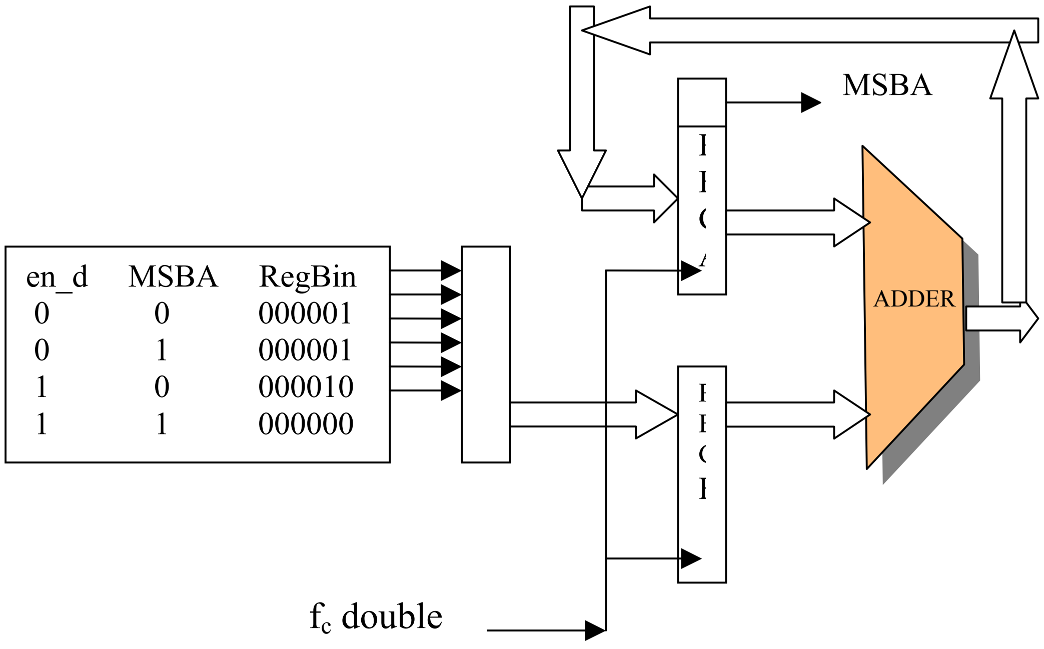
```
Name:   Value:
rst     0
fin     0
fc      0
en_d    0
```


MAX+plus II - e:\users\pkumar\proj\adpll - [adpll.scf - Waveform Editor]
MAX+plus II File Edit View Node Assign Utilities Options Window Help
86.0ns
200.0ns
400.0ns
600.0ns
800.0ns
1.0us
1.2us
1.4u

Ref: 86.0ns Time: 504.0ns Interval: 418.0ns

```
Name:   Value:
rst     1
fin     0
fc      1
en_d    0
```

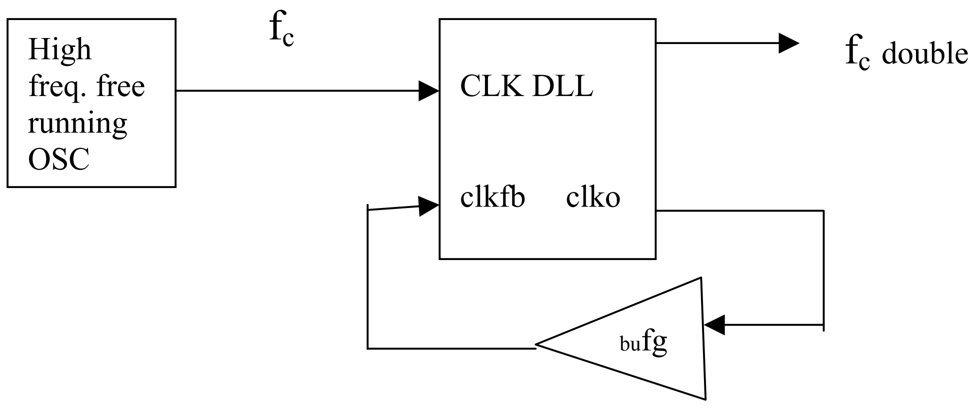
3.3 LOOP FILTER AND DCO CONTROL UNIT



en_d	MSBA	RegBin
0	0	000001
0	1	000001
1	0	000010
1	1	000000

This circuit simulates loop filter and DCO control unit. Registers A and B are of 6 bits each. (width of registers depends on frequencies of f_{in} and f_c . Here in this case
$f_c = 2^6 \times f_{in}$, so register width is 6)
Registers A and B are running on high frequency clock. MSB of Register A is our output clock locked to f_{in}. Reset value of Register B is "000001". If a positive edge of en_d comes and edge of MSBA does not come then value of register B is incremented for one clock cycle so as to get MSBA edge earlier in next turn. If a positive edge of en_d comes and edge of MSBA had came earlier then value of register B is decremented for one clock cycle so as to get MSBA edge later in next turn. In this way we get MSBA locked to f_{in}.

3.4 DCO CIRCUIT



This is the DCO circuit implemented by a high frquency free running oscillator and clock doubler circuit using DLL (Delay Locked Loop) of Xilinx FPGA. This is done to double the operating clock so as to have minimum jitter in output clock. It is observed that the jitter in output clock does not exceed the time period of this high frequency clock.
So we get less jitter when we double the clock using above circuit and use 2x f_c instead of simple f_c.

IMPLEMENTATION :

The above ADPLL was implemented using VHDL and synthesized in Xilinx FPGA Spartan-II series XC2S200-PQ208-5C. The ciruit was tested on PCB.
f_{in} was selected 2.048 MHz and f_c was selected 65.536 MHz. So 2x f_c became 131.072 MHz. Another case where f_{in} was 8.448 MHz was also tested (f_c remaing same).
Following are the VHDL code , VHDL test-bench and simulation results for the proposed ADPLL circuit :

3.5 VHDL CODE

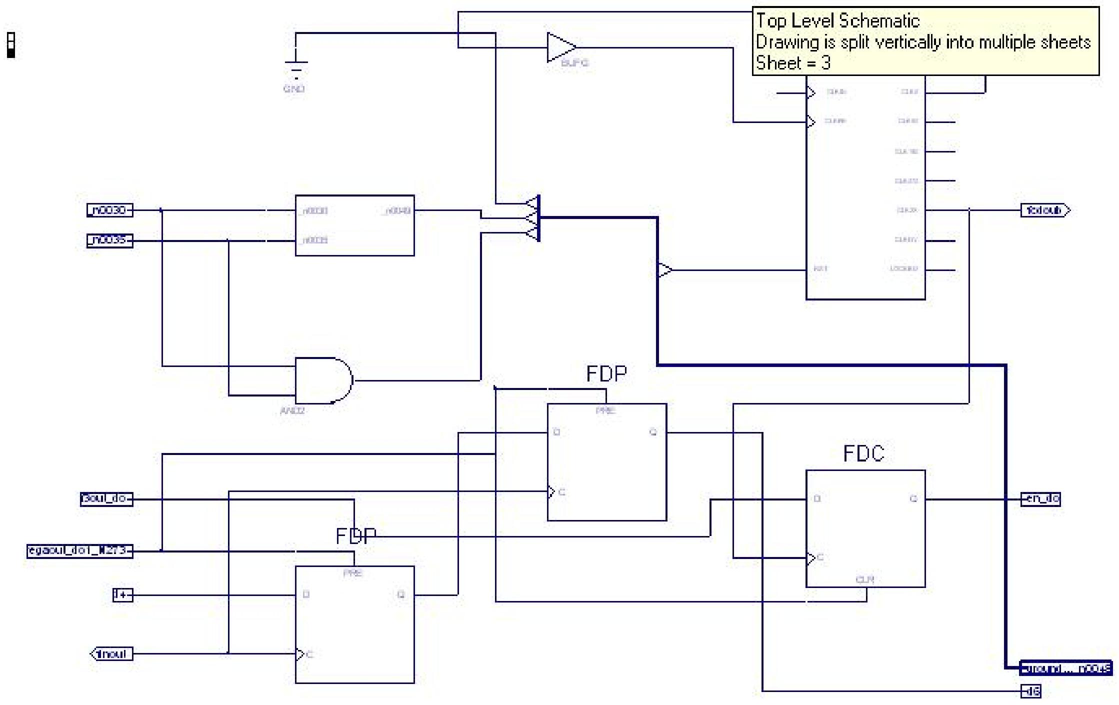
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_misc.all ;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity dproj is
port(
      fc          : in std_logic;
      fin         : in std_logic;
      rst         : in std_logic;
      f2mb        : out std_logic;
      f2mb_do     : out std_logic;
      f8mb        : out std_logic;
      rec_clk     : out std_logic;
      fcout       :  out std_logic;
      finout      : out std_logic;
      fcdoub      : out std_logic;
      scrdatao    : out std_logic
       );
end dproj;

architecture RTL of dproj is

component clkdll port
(
 clkin            :     in std_logic;
 clkfb            :     in std_logic;
 rst              : in std_logic;
 clk0       :     out std_logic;
 clk90            :     out std_logic;
 clk180      :  out std_logic;
 clk270     :     out std_logic;
 clk2x            :     out std_logic;
 clkdv            :     out std_logic;
 locked           :     out std_logic
);
end component;

component bufg port
(
      i     :     in std_logic;
      o     :     out std_logic
);
end component;

signal f1out        : std_logic;
signal f2out        : std_logic;
signal f3out        : std_logic;
signal en           : std_logic;
signal en_d         : std_logic;
signal msba         : std_logic;
signal regbin       : std_logic_vector(4 downto 0);
signal regaout      : std_logic_vector(4 downto 0);
signal regbout      : std_logic_vector(4 downto 0);
signal adderout     : std_logic_vector(4 downto 0);

signal fcdouble     : std_logic;
signal clk_out      : std_logic;
signal clkfb        : std_logic;
signal ground       : std_logic;

signal f1out_do      : std_logic;
signal f2out_do      : std_logic;
signal f3out_do      : std_logic;
signal en_do         : std_logic;
signal en_d_do       : std_logic;
signal msba_do       : std_logic;
signal regbin_do     : std_logic_vector(5 downto 0);
signal regaout_do    : std_logic_vector(5 downto 0);
signal regbout_do    : std_logic_vector(5 downto 0);
signal adderout_do   : std_logic_vector(5 downto 0);

signal msba_do8        : std_logic;
signal regbin_do8      : std_logic_vector(8 downto 0);
signal regaout_do8     : std_logic_vector(8 downto 0);
signal regbout_do8     : std_logic_vector(8 downto 0);
signal adderout_do8    : std_logic_vector(8 downto 0);

signal scrdata          : std_logic;
signal d1,d2,d3,d4,d5,d6,d7 : std_logic;
signal f1out_do1        : std_logic;
signal f2out_do1        : std_logic;
signal f3out_do1        : std_logic;
signal en_do1           : std_logic;
signal en_d_do1         : std_logic;
signal msba_do1         : std_logic;
signal regbin_do1       : std_logic_vector(5 downto 0);
signal regaout_do1      : std_logic_vector(5 downto 0);
signal regbout_do1      : std_logic_vector(5 downto 0);
signal adderout_do1     : std_logic_vector(5 downto 0);


begin

dll: clkdll port map
(
      clkin       => fc ,
      clkfb       => clkfb,
      rst               => ground ,
      clk0             => clk_out,
      -- clk_90         => open,
      -- clk_180        => open,
      -- clk_270        => open,
      clk2x       => fcdouble
      -- clk_dv         => open,
      -- locked         => open
);

fbclk :  bufg port map
(
      i     =>    clk_out,
      o     =>    clkfb
   );

process(en, fin, rst)
begin
        if (rst = '0'or en = '1') then
            f1out  <= '0';
        elsif fin'event and fin = '1' then
             f1out <= '1';
        end if;
end process;

process(f1out, fc, rst)
begin
        if rst = '0' then
               f2out <= '0'; f3out <= '0'; en <= '0' ;
         elsif fc'event and fc = '1' then
           f2out <= f1out ; f3out <= f2out ; en <= f3out ;
        end if;
end process;


process(en_d,fc, rst)
begin
        if rst = '0' then
         regaout <= "00000"; regbout <= "00001";
        elsif fc'event and fc = '1' then
            regaout <= adderout;
            regbout <= regbin ;
        end if;
```

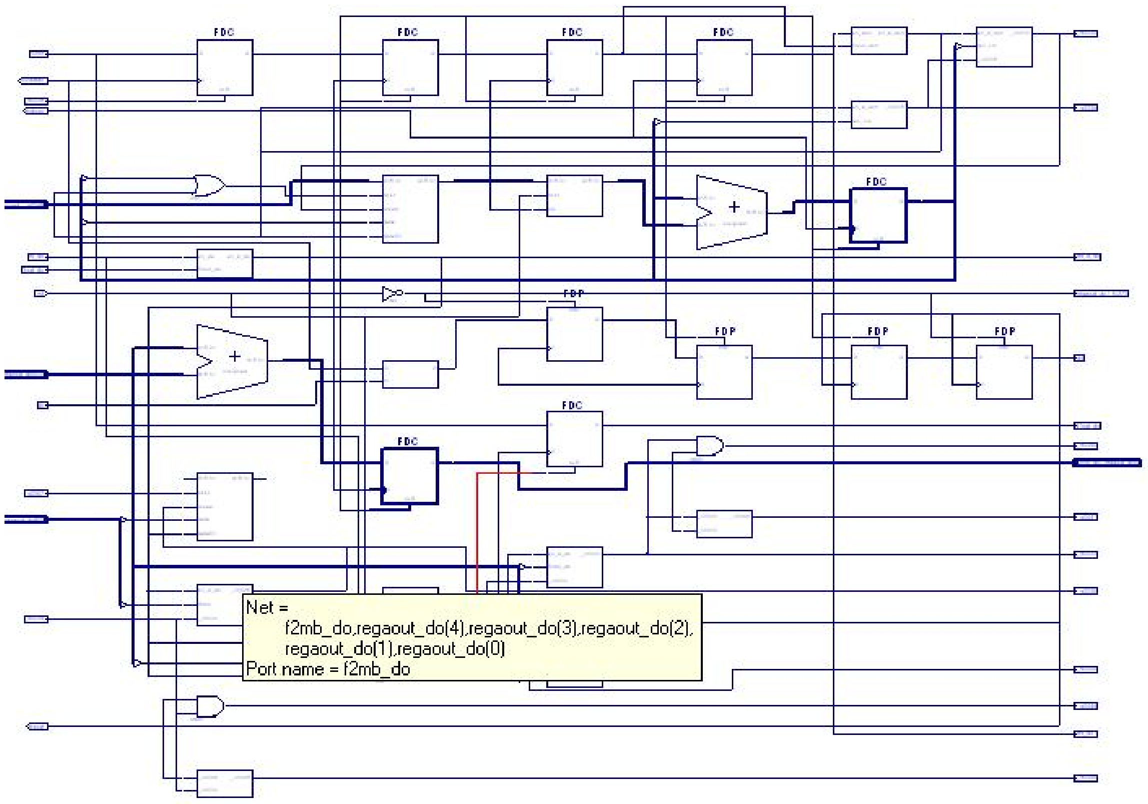
```
    if ( en_d='0' and msba='0') then regbin <="00001" ;
 elsif ( en_d='0' and msba='1') then regbin <="00001" ;
 elsif ( en_d='1' and msba='0') then regbin <="00010" ;
 elsif ( en_d='1' and msba='1') then regbin <="00000" ;
 end if;

end process;

msba <= regaout(4) ;
adderout <= regaout + regbout ;
f2mb <= msba ;
fcout <= fc;
finout <= fin;
en_d <= (not en) and f3out;
fcdoub <= fcdouble;
--------
process(en_do, fin, rst)
begin
        if (rst = '0'or en_do = '1') then
            f1out_do  <= '0';
        elsif fin'event and fin = '1' then
             f1out_do <= '1';
        end if;
end process;
process(f1out_do, fcdouble, rst)
begin
        if rst = '0' then
            f2out_do <= '0'; f3out_do <= '0';
            en_do <= '0' ;
         elsif fcdouble'event and fcdouble = '1' then
           f2out_do <= f1out_do ; f3out_do <= f2out_do ;
           en_do <= f3out_do ;
        end if;
end process;
process(en_d_do,fcdouble, rst)
begin
        if rst = '0' then
         regaout_do <= "000000"; regbout_do <= "000001";
        elsif fcdouble'event and fcdouble = '1' then
            regaout_do <= adderout_do;
            regbout_do <= regbin_do ;
        end if;
    if ( en_d_do='0' and msba_do='0') then regbin_do <="000001" ;
 elsif ( en_d_do='0' and msba_do='1') then regbin_do <="000001" ;
 elsif ( en_d_do='1' and msba_do='0') then regbin_do <="000010" ;
 elsif ( en_d_do='1' and msba_do='1') then regbin_do <="000000" ;
end if;
end process;

msba_do <= regaout_do(5) ;
adderout_do <= regaout_do + regbout_do ;
f2mb_do <= msba_do ;
en_d_do <= (not en_do) and f3out_do;
ground <= '0' ;

process(en_d_do,fcdouble, rst)
begin
      if rst = '0' then
       regaout_do8 <= "000000000"; regbout_do8 <= "000100001";
        elsif fcdouble'event and fcdouble = '1' then
            regaout_do8 <= adderout_do8;
            regbout_do8 <= regbin_do8 ;
        end if;
    if ( en_d_do='0' and msba_do8='0') then regbin_do8 <="000100001" ;
 elsif ( en_d_do='0' and msba_do8='1') then regbin_do8 <="000100001" ;
 elsif ( en_d_do='1' and msba_do8='0') then regbin_do8 <="000100010" ;
 elsif ( en_d_do='1' and msba_do8='1') then regbin_do8 <="000100000" ;
 end if;

end process;

msba_do8 <= regaout_do8(8) ;
adderout_do8 <= regaout_do8 + regbout_do8 ;
f8mb <= msba_do8 ;

----
process(d1,d2,d3,d4,d5,d6,d7,rst,fin,scrdata)
begin
        if rst = '0' then
        d1<='1'; d2<='1'; d3<='1'; d4<='1'; d5<='1';
            d6<='1'; scrdata<='1';
         elsif fin'event and fin = '1' then
        scrdata <= d6; d6<=d5; d5<=d4; d4<=d3; d3<=d2;
        d2<=d1; d1<=d7;
        end if;
end process;
d7<= d6 xor scrdata ;
scrdatao <= scrdata;

process(en_do1, fin, rst)
begin
        if (rst = '0'or en_do1 = '1') then
            f1out_do1  <= '0';
        elsif scrdata'event and scrdata = '1' then
             f1out_do1 <= '1';
        end if;
end process;
process(f1out_do1, fcdouble, rst)
begin
        if rst = '0' then
               f2out_do1 <= '0'; f3out_do1 <= '0'; en_do1 <= '0' ;
         elsif fcdouble'event and fcdouble = '1' then
           f2out_do1 <= f1out_do1 ; f3out_do1 <= f2out_do1 ;
           en_do1 <= f3out_do1 ;
        end if;
end process;
process(en_d_do1,fcdouble, rst)
begin
        if rst = '0' then
         regaout_do1 <= "000000"; regbout_do1 <= "000001";
        elsif fcdouble'event and fcdouble = '1' then
            regaout_do1 <= adderout_do1;
            regbout_do1 <= regbin_do1 ;
        end if;
    if ( en_d_do1='0' and msba_do1='0') then regbin_do1 <="000001" ;
 elsif ( en_d_do1='0' and msba_do1='1') then regbin_do1 <="000001" ;
 elsif ( en_d_do1='1' and msba_do1='0') then regbin_do1 <="000010" ;
 elsif ( en_d_do1='1' and msba_do1='1') then regbin_do1 <="000000" ;
end if;
end process;
msba_do1 <= regaout_do1(5) ;
adderout_do1 <= regaout_do1 + regbout_do1 ;
rec_clk <= msba_do1 ;
en_d_do1 <= (not en_do1) and f3out_do1;


end RTL;

---------------------------------------------------------
```

3.6 VHDL TEST-BENCH

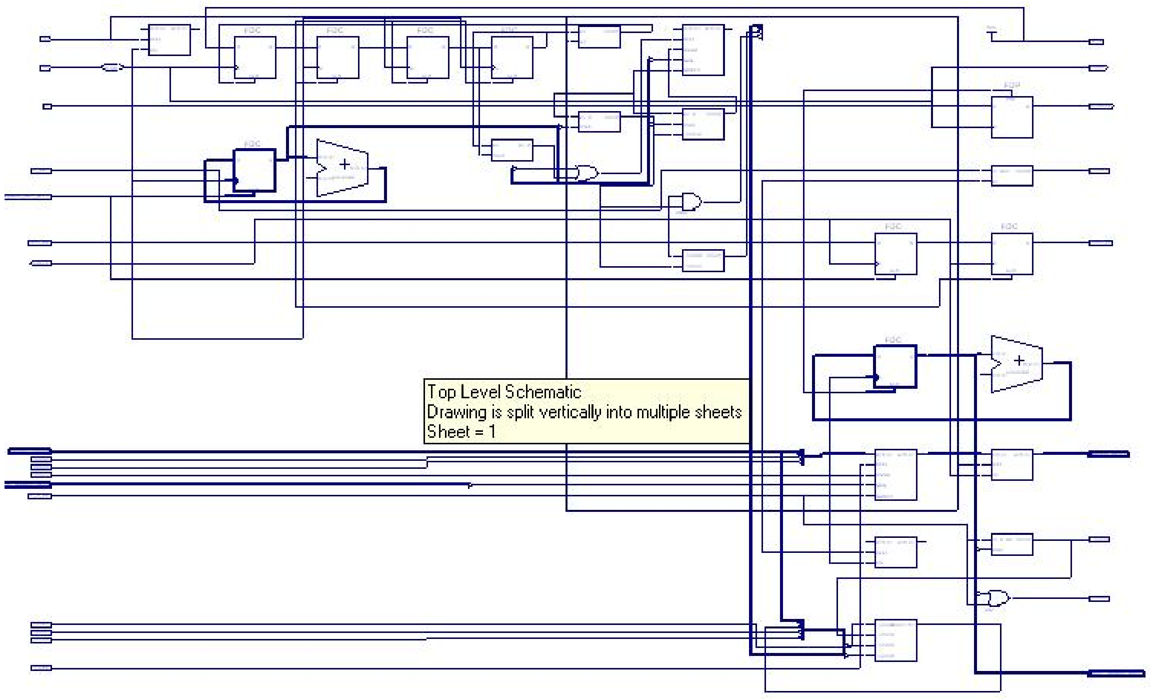
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_misc.all ;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity tb_dproj is
end tb_dproj;

architecture test of tb_dproj is

component dproj port

 (     fc           : in std_logic;
       fin          : in std_logic;
       rst          : in std_logic;
       f2mb         : out std_logic;
       f2mb_do      : out std_logic;
       f8mb         : out std_logic;
       rec_clk      : out std_logic;
       fcout        :  out std_logic;
       finout       : out std_logic;
       fcdoub       : out std_logic;
       scrdatao     : out std_logic
        );
end component;

signal      fc          :  std_logic;
signal      fin         :  std_logic;
signal      rst         :  std_logic;
signal      f2mb        :  std_logic;
signal      f2mb_do     :  std_logic;
signal      f8mb        :  std_logic;
signal      rec_clk     :  std_logic;
signal      fcout       :   std_logic;
signal      finout      :  std_logic;
signal      fcdoub      :  std_logic;
signal      scrdatao    :  std_logic;


begin

dut :  dproj port map

(     fc           => fc ,
      fin          => fin ,
      rst          => rst,
      f2mb         => f2mb,
      f2mb_do      => f2mb_do,
      f8mb         => f8mb,
      rec_clk      => rec_clk,
      fcout      => fcout ,
      finout       => finout,
      fcdoub       => fcdoub,
      scrdatao     => scrdatao
       );

           process
            begin
            rst <= '0' ;
            wait for 50 ns;
            rst <= '1' ;
            wait  ;
            end process;


            process
            begin
               fin  <= '0';
             wait  for 125 ns;
               fin  <= '1';
             wait  for 125  ns;
            end process;

           process
           begin
             fc  <= '0';
           wait  for 15.625  ns;
             fc  <= '1';
           wait  for 15.625 ns ;
          end process;

end test;

configuration cfg_tb_dproj of tb_dproj is
  for test
  end for;
end cfg_tb_dproj;
```

3.7 SCHEMATICS :

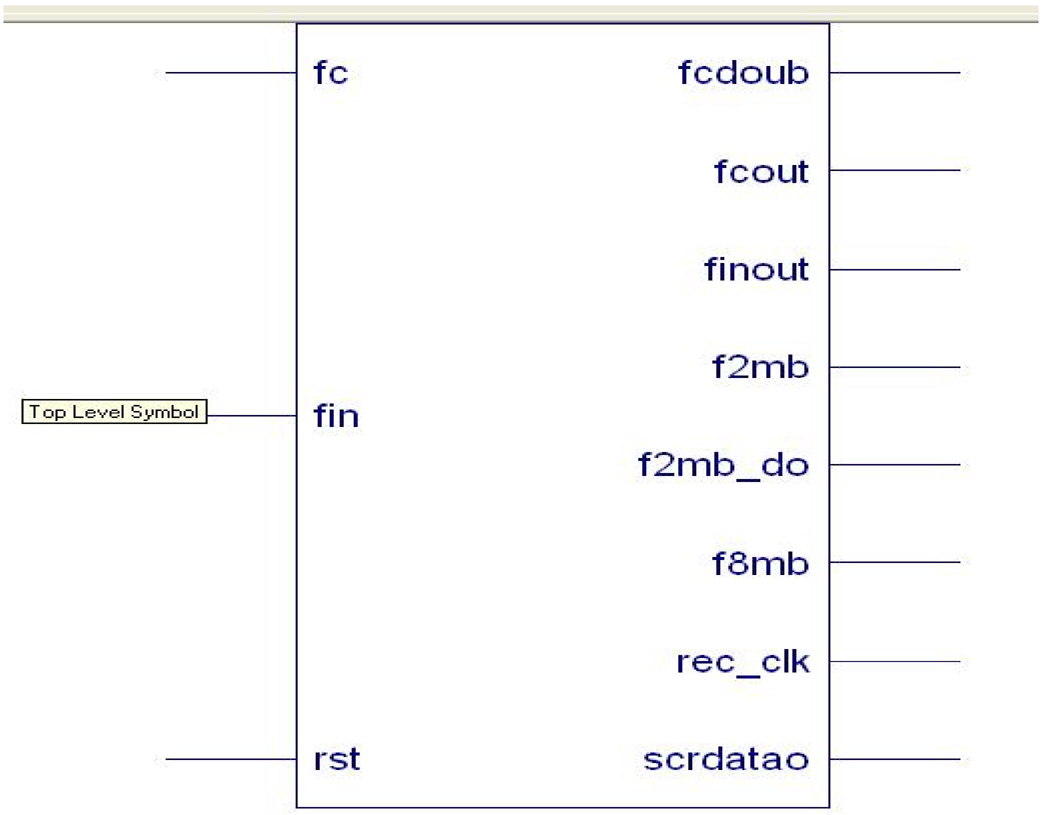




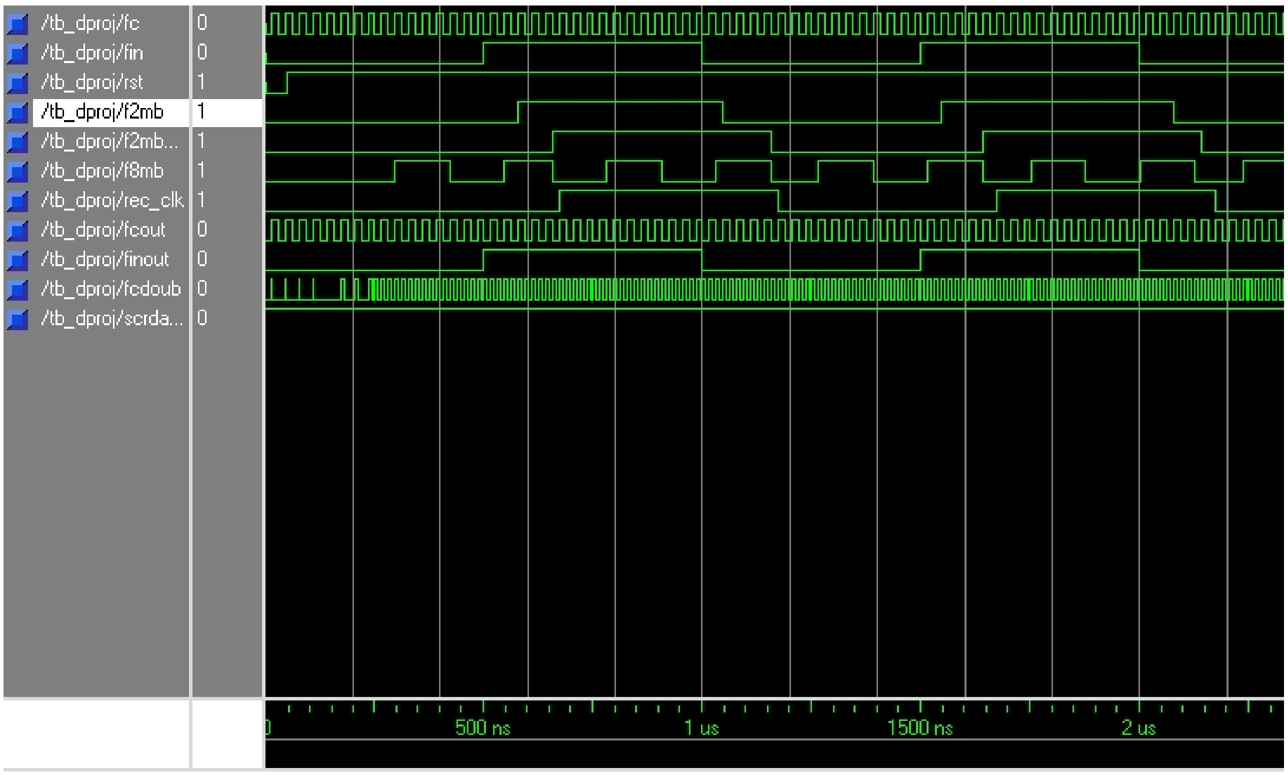
Net =
f2mb_do,regaout_do(4),regaout_do(3),regaout_do(2),
regaout_do(1),regaout_do(0)
Port name = f2mb_do

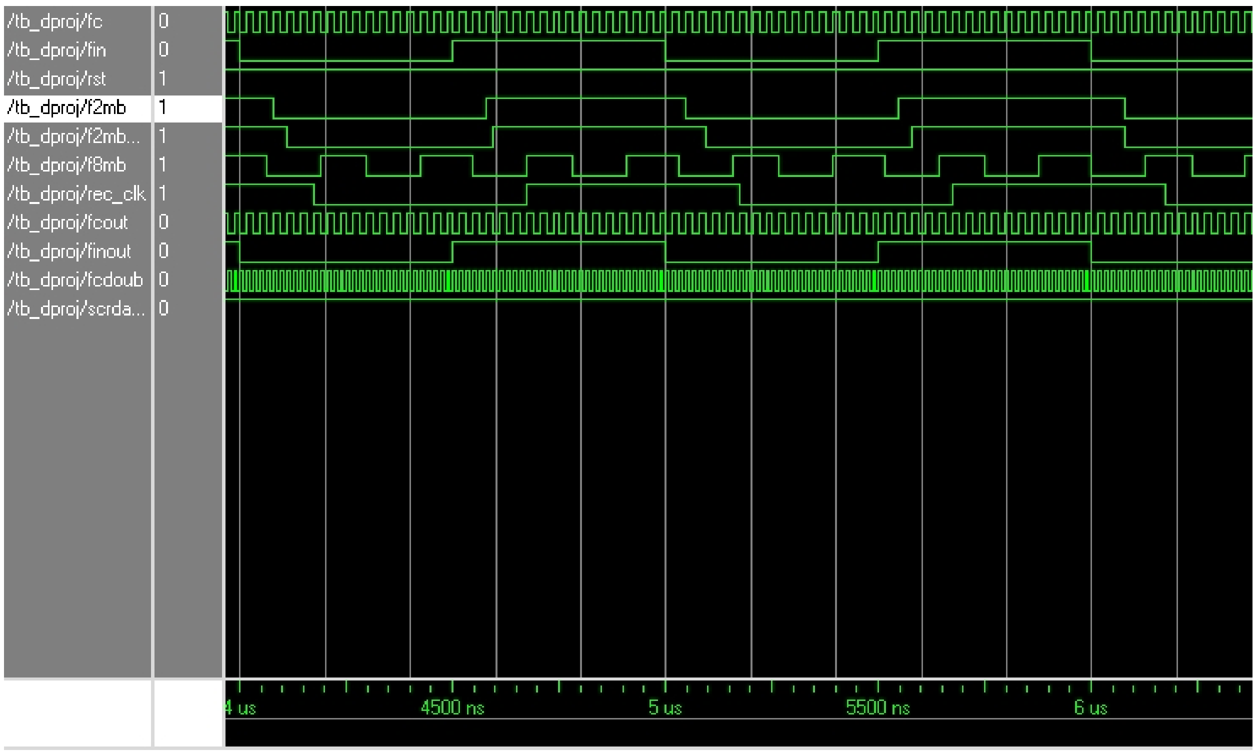


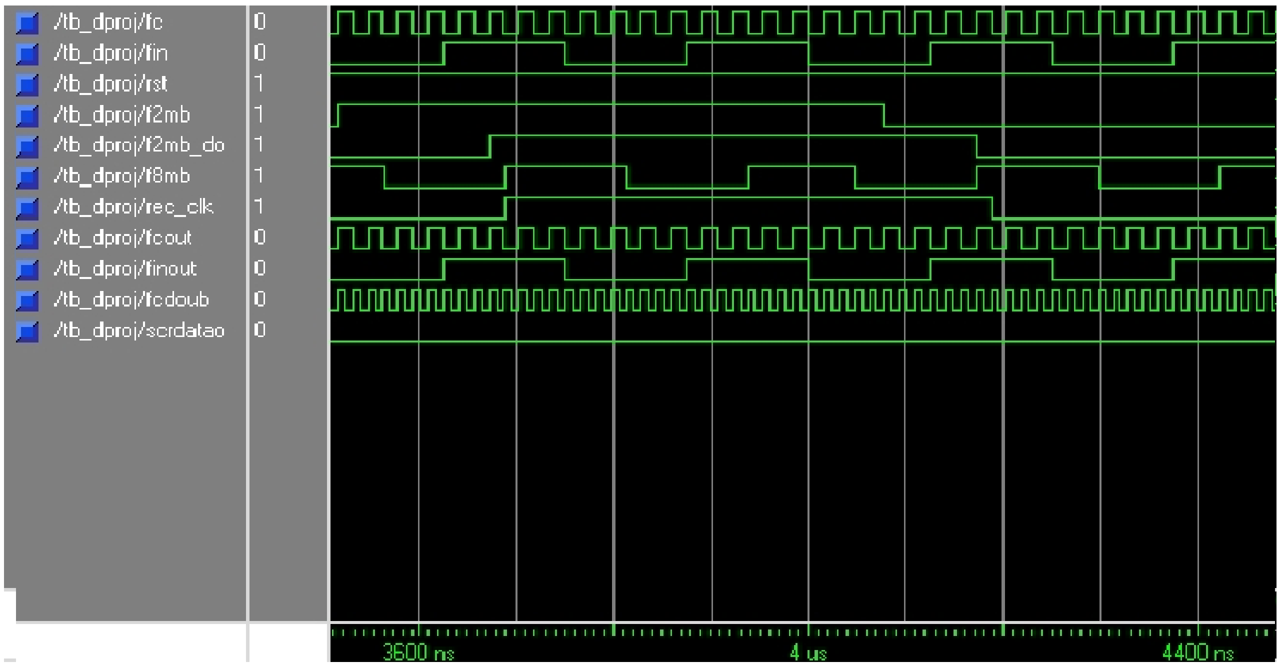
Top Level Schematic
Drawing is split vertically into multiple sheets
Sheet = 1

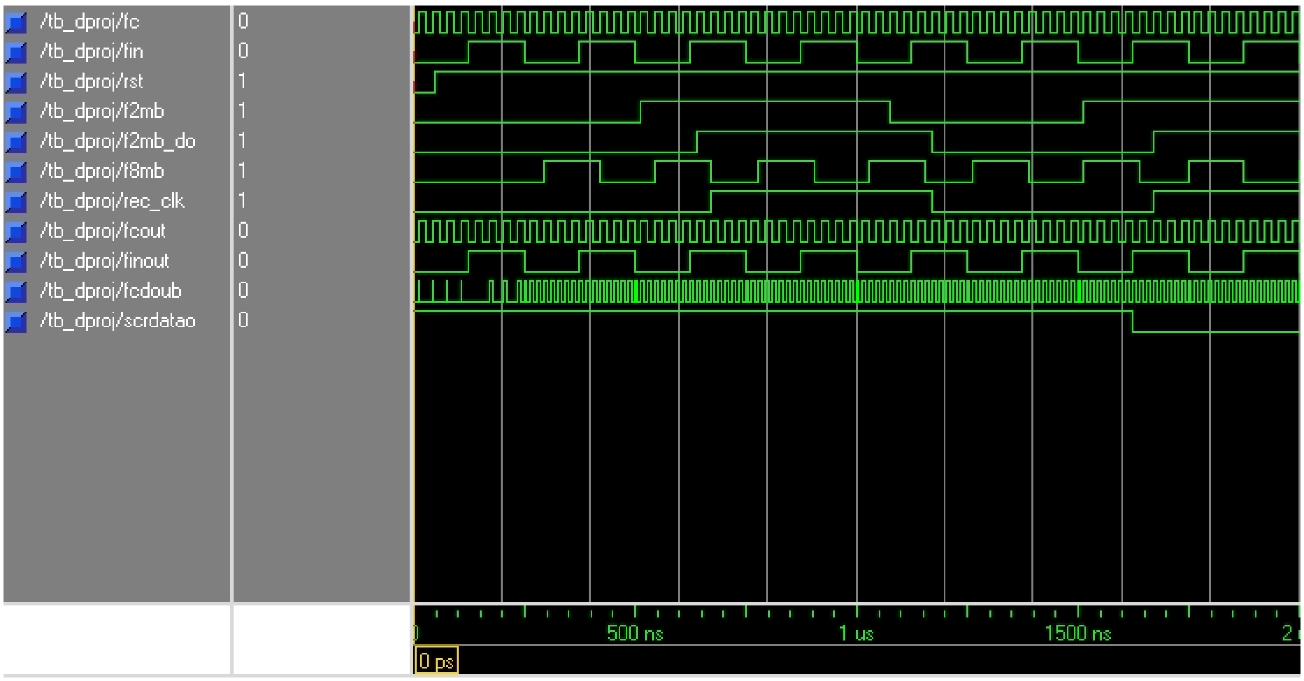


SIMULATIONS :

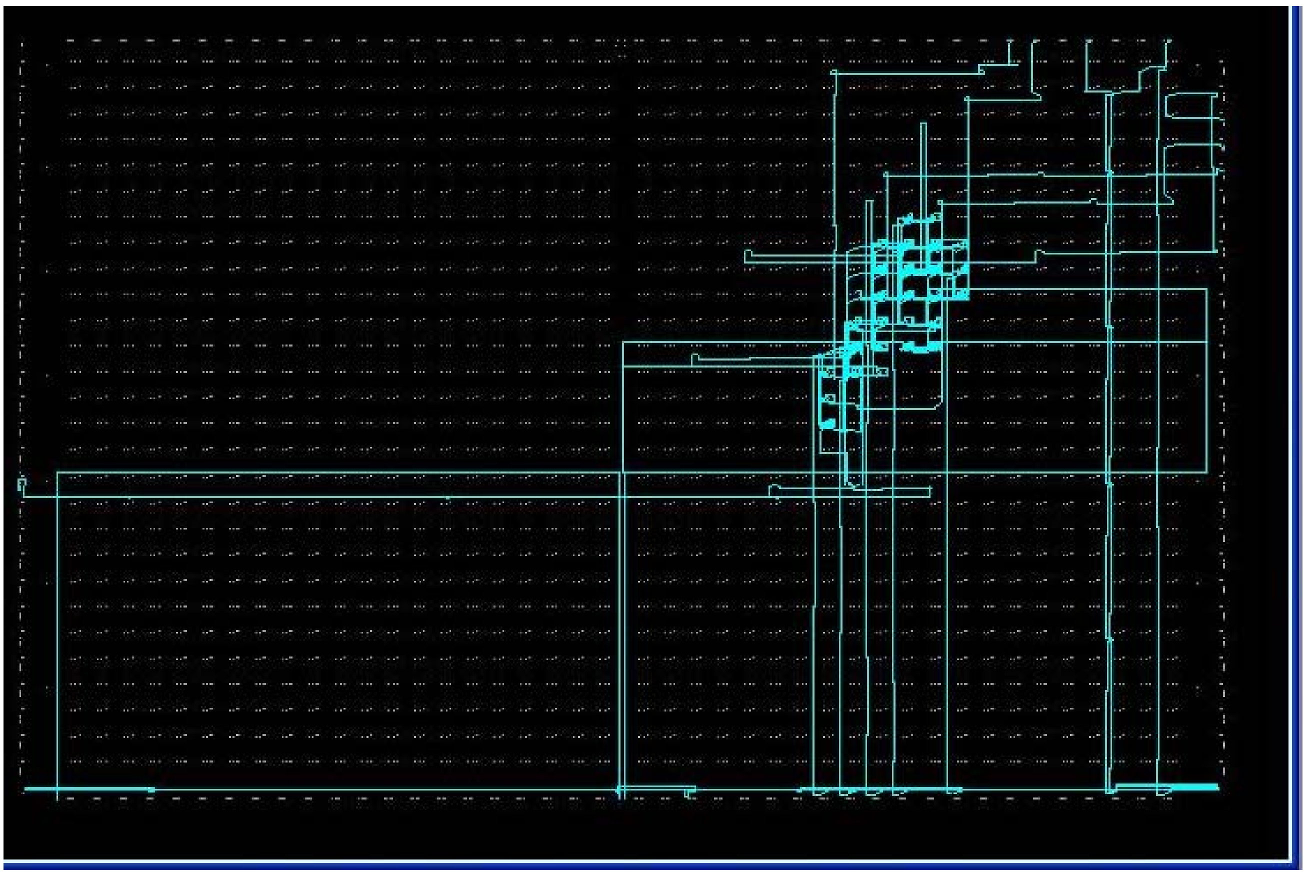



/tb_dproj/fc 0
/tb_dproj/fin 0
/tb_dproj/rst 1
/tb_dproj/f2mb 1
/tb_dproj/f2mb... 1
/tb_dproj/f8mb 1
/tb_dproj/rec_clk 1
/tb_dproj/fcout 0
/tb_dproj/finout 0
/tb_dproj/fcdoub 0
/tb_dproj/scrda... 0
4 us
4500 ns
5 us
5500 ns
6 us


/tb_dproj/fc 0
/tb_dproj/fin 0
/tb_dproj/rst 1
/tb_dproj/f2mb 1
/tb_dproj/f2mb_do 1
/tb_dproj/f8mb 1
/tb_dproj/rec_clk 1
/tb_dproj/fcout 0
/tb_dproj/finout 0
/tb_dproj/fcdoub 0
/tb_dproj/scrdatao 0
3600 ns
4 us
4400 ns



SYNTHESIZED DESIGN IN FPGA:



CONSTRAINTS FILE :

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "f2mb"  LOC = "P164"  ;
NET "f2mb_do"  LOC = "P163"  ;
NET "f8mb"  LOC = "P152"  ;
NET "fc"  LOC = "P77"  ;
NET "fcdoub"  LOC = "P160"  ;
NET "fcout"  LOC = "P161"  ;
NET "fin"  LOC = "P80"  ;
NET "finout"  LOC = "P162"  ;
NET "rec_clk"  LOC = "P150"  ;
NET "rst"  LOC = "P30"  ;
NET "scrdatao"  LOC = "P151"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

4. SWITCHING

4.1 SWITCHING BACKGROUND

The use of digital techniques has always been essential in providing universal telecommunication service: whether it was the digital dexterity of the operators selecting a jack to complete a call, or the digits placed into a switching system to control the remote and automatic selection of a telecommunication terminal or path. In recent years voice signals that constitute the messages in some telecommunication systems have been digitized. This made possible the distortion less robust transmission of signals representing the voice. "Digital Transmission" as it is called, has not only made possible high speed communication between terminals offering discrete signals, such as alarm, computer, and written communication, but has stimulated a new era in switching where the path through the switching center is also designed to pass digitized signals.

Based on this newer application of the "digital" techniques, the term "digital switching" has now come into prominent usage. However, many who use the term mean not only the network to switch digital signals, but also the digital control process that has been a characteristic of switching since its origin.

4.2 ELECTRONICS IN SWITCHING

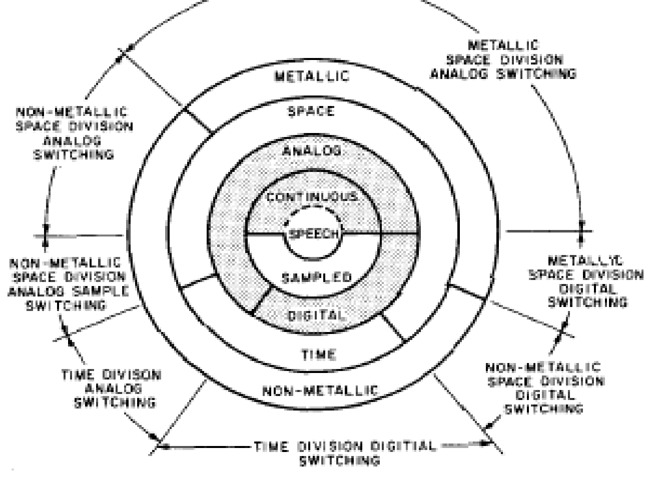
Switching system designers dreamed from early 1920's of harnessing the advantages of speed of electronics, the technology then being used to advantage in extending transmission capability in distance and in capacity. But since for many digital techniques many logic elements were required, the number, reliability, and power required for such information processing made it impractical to use hot cathode electronic technology. A single electromagnetic relay represents as many as 24 logic elements and has a low power duty cycle while a vacuum tube consumed continuos power. This is still one of the drawbacks of the large scale use of electronics, but with the reduced size of semi-conductor devices has come a reduction of power by several orders of magnitude.

Even prior to the invention of the transistor, the principle of time-sharing was recognized as a way of taking advantage of the speed of electronics. Systems using vacuum tubes for logic operations were applied to computer, time division transmission and switching. The sampling of speech led H. Nyquist to postulate a criterion for frequency for sampling and accurately reproducing a given bandwidth signal. Sample from different sources could then "time share" the transmission medium if its bandwidth was great enough. The time sharing of a communication medium to transmit samples of varying amplitude captured the imagination of many electronic experts fresh from their World

War II experiences in radar and the like and who were looking for an opportunity to apply their skills to such civilian pursuits as telecommunications.
The World War II effort also stimulated more serious consideration of serial transmission of on and off or “digital” signals first proposed in 1926 in connection with picture transmission. With digital transmission the samples of the signal amplitude were coded as one method to make feasible speech secrecy. In a sense this was the beginning of digital switching because for the first time the digitized speech samples were processed for than other digital/analog conversion and retransmission. Special electron tubes, also from these wartime efforts, gave rise to electronic storage, scanning and coding.
Prior to the concept of time sharing, all circuit switching networks established continuos and separate two-way transmission paths. This is known as “space division”. With sampled speech time sharing, known initially as “time sharing”, and now as “time division” switching became possible.
With the invention of the transistor, electronic logic for switching controls became feasible. This meant that several competing common controls were necessary to serve offered calls in real time with relay logic, only a single control taking advantage of the higher speed of electronics would be adequate as the common control for a switching system. Shortly thereafter, the ideas of using bulk electronic memory with random access and electronic semiconductor logic were combined to produce3 the first real time programmed common control.

4.3 DEFINITIONS

A “digital switching system” is the one in which signals representing messages aretransmitted through the switch in digital form.
As shown in figure all speech signals start out as continuos analog signals – from sound to varying electrical signals. If speech signals reach a digital switch in analog form they must first be converted by sampling techniques to any one of several digital forms. Usually for central office switching the form chosen matches the form used by most transmission systems, that is 8-bit per sample pulse code modulation (PCM).



4.4 DIGITAL SWITCHING FUNDAMENTALS

There are many ways to switch speech signals that have been digitized. An electro-mechanical switch may be used to interconnect digital transmission facilities, a function required for example to use spare facilities when trouble develops in another line on the same route, known as “span switching”. Digitized signals may be passed through an electro-mechanical switch (such as crossbar). This is a metallic space division digital circuit switching system. However, even to do this efficiently requires some new concepts that are useful in understanding the fundamentals of any digital switch.

To be efficiently transmitted both outside and generally within a switching office, coded digital samples are usually time multiplexed. As a result, a plurality of channels is available for transmitting messages simultaneously over one transmission channel. Here, as in frequency multiplex for analog trans-mission, the greater the useful bandwidth, the greater the capacity. In time division multiplex (TDM) each position for a sample, or channel, is known as a “time slot”. Collectively, one sample from each channel constitutes a “frame” of signals. The individual pulses of a sample are usually transmitted
serially.

Generally, a switching system serves several TDM lines. To switch calls then requires digital signals from individual time slots on one line to be placed in the same or different time slots on other lines. To efficiently use a digital line requires that most time slots be assigned and used. Under these circum-stances it is quite likely that the signals from one time slot would have to be placed into a different time slot on another line. This “shifting” or “interchanging” of time slots isessential to efficient time division switching as it relates to integrated digital transmission and switching. This principle was early recognized by Messrs. H. hose and J. P. Runyon in 1960.

If metallic space division switching is used for switching digital signals, time buffering and TSI are required for all time slots in other lines to be accessible. Placing and later removing coded speech samples from buffers or memories inherently introduces delay in transmission which must be kept as short as possible. Generally each memory or time stage *(T)* through which a sample passes may introduce a maximum delay of almost two frames since it is necessary to store the samples for one frame while reading out from another section of memory the samples from the preceding frame.

Memory may be provided in any of several forms. In the early days of digital switching variable delay lines were popular. They received and delivered signals serially. Now
most memories use integrated circuit technology with parallel access.

Switching between the same time slots on different digital transmission lines may use a common multiplex and time stage or a form of space division switching. When switching is purely sequential, it is known as multiplexing. In a multi-TI-line system channels enter the system time divided on each line and separated in space (space division) from different lines. The time and space divided channels are then converted to time division by multiplexing. In this process concentration is possible with more channels as sources than there are time slots. In some systems to improve the traffic performance expansion it is also introduced by serving fewer channels than time slots. Multiplexers operate in a fixed order and provide for concentration by skipping channels not being served.

The rate at which the multiplex stage loads a time division internal link is an indication of the degree to which the electronic technology is being pressed. Within a modern digital
TDM switching system the digitized samples are read into and out of memory in parallel. Since the bits of the samples to and from lines are serial, buffers are required for serial to parallel (and reverse) conversion. In some systems to increase the capacity with a given technology the samples are passed between stages within the system on parallel buses or
a combination of serial and parallel.

Multiplexing is generally not considered to be a switching network function, but when strictly' considered, including the TSI memory access, it and de-multiplexing are functions that enable digital TDM networks to be composed of only *T* stages. In general, multiplexing is used for space to time conversion.

Unlike space division switching for analog signals where connections are maintained continuously for each message, space division switches for time division networks are used to interconnect (permute) time division transmission lines for each time slot period. This has become known as "time multiplex switching" (TMS), but for simplicity, it is in the context of time division switching known as "space" (S) switching. These are multiplex stages with random rather than fixed order of gating with access to a common set of inlets or outlets. Furthermore, they are usually arranged so that each channel is gated in each time slot. As indicated earlier, systems with only S stages are inefficient from a traffic standpoint. However, using combinations of S and *T* stages is a popular way of increasing network capacity. In particular, the S stages facilitate system growth.

From a traffic handling point of view, the *T* stages are non-blocking and the S stages may be engineered to be non-blocking. Non-blocking S stages act non-blocking for all time slots. Therefore it is relatively inexpensive for the non-concentrating portions of TDM switching systems to be made non-blocking. The total number of input (and therefore output) time slots is equal to the number of independent time slots provided within the system. This total is equal to tm where m equals the number of buses or highways and t equals the number of time slots per bus. Each TSI memory section has at least as many addresses as there are time slots. Single and three or more switching stages of square switches are also non-blocking by rearrangement. Generally to insure this characteristic without rearrangement, more internal time slots are provided. In addition to providing more time slots, the inputs from several sources that are multiplexed together may be decorrelated so that heavy traffic or failure from a particular digital line will be spread throughout the remainder of the switch.

In any switching network the connection relationships are stored in memory, In space division electromechanical systems this memory is usually inherent in the device states. In time division switching the connection relationship is in cyclical bulk memory. These memories are loaded for each connection and provide the TSI memory read out addresses and the TMS connection permutations.

Concentration, that is, more input channels than output time slots, may be included in the multiplexing process. Further multiplexing is usually employed to bring together several time division sources, sometimes with a slight expansion (more output time slots than total input time slots). Also, in multiplexing it is not unusual to decorrelate the time slots from each of the several sources. Since there may be different pulse transmission rates and the pulse may come from unrelated sources, intermediate buffer storage of coded samples is often required before multiplexing.

All multiplexing is in preparation for the actual switching. Concentration, expansion, and decorrelation are switching functions. Switching provides for the association of input time slots with output time slots for all or any portion of the network. To do this requires a random access control memory that contains the required input-output time slot associations regardless of how the switching is accomplished.

The association of coded samples in given input time slots with assigned output time slots may occur in either of two ways. One is by storing the samples. This implies a time *(T)*
delay to affect a change in time slot. The other way is by establishing a direct connection, that is switching (S), directly to an output link or line in the same time slot without delay.

When viewed from an input or output this high speed time slot switching is random access multiplexing, with a restriction that no more than one input or output may be simultaneously connected. In some systems this is known as "time multiplex switching (TMS)".

Generally switching networks are composed of ***T*** stages which are non-blocking or combinations of ***T*** and S stages in any order. Network types are denoted by the order of the stages, such as ***T*, *TST,*** or ***STS.*** For specific systems some authors indicate the actual number of stages by repeated use of the letters, such as ***TSSSST*** In larger systems to insure service continuity in the presence of device failure, part or all of the network is made redundant.

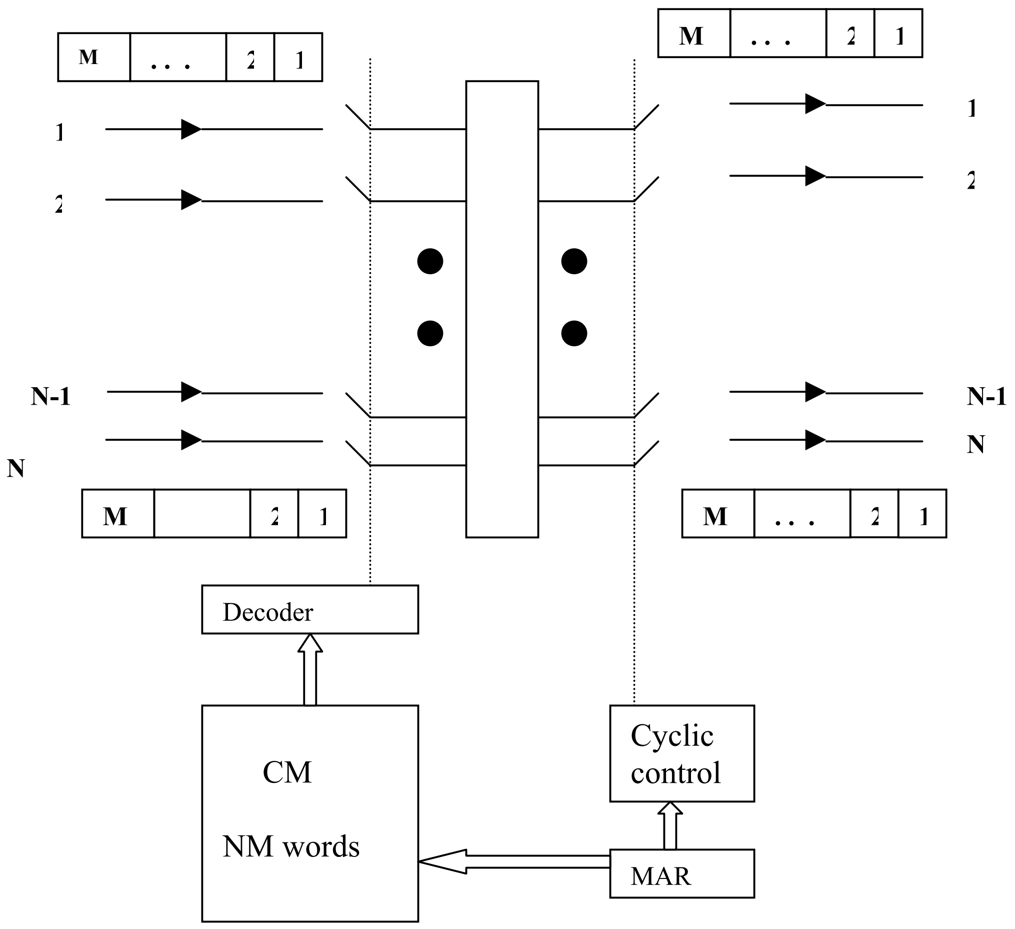
An important area that has been explored in depth deals with the subject of pulse timing or synchronization, to ensure that switching centers and transmission lines are able to operate together so that frames of pulses will not be lost or repeated. Most important in this process is the distribution of standard frequencies from which all pulses generated in the network may be derived. Pulses derived from a national-wide standard trigger or drive the transmission and switching system pulse generators, known as "clocks". In some systems the clocks are

synchronized with pulses derived by averaging the pulse rates of all incoming digital transmission lines.

4.5 TIME MULTIPLEXED SPACE SWITCHING

In time division switches where an inlet or an outlet corresponds to single subscriber line with one speech sample appearing every 125μs on the line. Such switches are used in local exchanges. We now consider switches that are required in transit exchanges. Here, inlets and outlets are the trunks which carry time division multiplexed data streams. We call such switches **time multiplexed switches**.

A time multiplexed time division space switch is shown in figure. There are N incoming trunks and N outgoing trunks, each carrying a time division multiplexed stream of M samples per frame. Each frame is of 125 μs time duration. In one frame time, a total number of MN speech samples have to be switched. One sample duration, 125/M microseconds, is usually referred to as a time slot. In one time slot, N samples are switched. In case of output controlled switch (which is our present case), the output is cyclically scanned. There is a 1-to-M relationship between the outlets and the control memory locations, i.e., there are M locations in the control memory corresponding to each outlet.



The control memory has MN words, If we view the control memory as M blocks of N words each, a location address may be specified in a two dimensional form, (i,j), where i is the block address and j is the word within the block. We have $1 \le i \le M$ and $1 \le j \le N$. The block address i corresponds to the time slot i and the word address j to outlet j. The first N locations of control memory correspond to the first time slot, the next N locations, i.e., Locations N+1 to 2N when addressed linearly, or locations (2,1) to (2,N) when addressed in a two dimensional form, correspond to the time slot 2 and so on. Therefore, if location (i,j) contains an inlet address k, it implies that inlet k is connected to the outlet j during the time slot i . The number of trunks that can be supported on this switch is given by

$N= 125/ Mt_s$

Where t_s is the switching time including memory access time per inlet-outlet pair. The cost of the switch is estimated as

C = No. of switches + No. of memory words

= 2N + MN

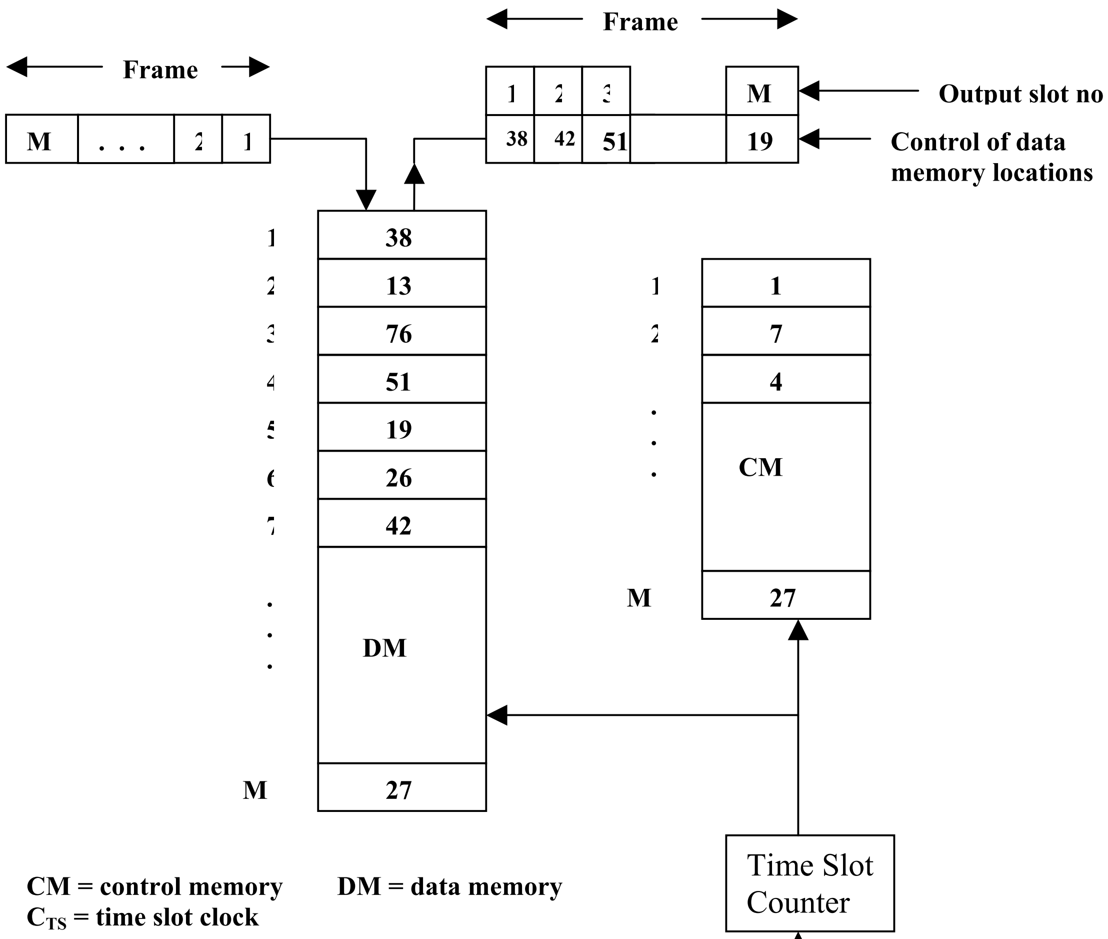
The cost of an equivalent single-stage space division network is $(MN)^2$.

4.6 TIME MULTIPLEXED TIME SWITCH

Unlike time multiplexed space switches, time multiplexed time switches permit time slot interchange (TSI) of sample values. In TSI, a speech sample input during one time slot may be sent to the output during a different time slot. Such an operation necessarily implies a delay between the reception and transmission of a sample. We illustrate the principle of TSI by considering a time switch with one incoming trunk and one outgoing trunk as shown in figure. M channels are multiplexed on each trunk. The switch is organized in the sequential write/random read fashion. The time slot duration is given by

$t_{TS} = 125/M$

The time slot clock runs at the time slot rate, i.e. at the rate of one pulse every 125/M microseconds. The time slot counter is incremented by one at the end of each time slot. The contents of the counter provides location addresses for the data memory and the control memory. Data memory and control memory access takes place simultaneously in the beginning of the time slot. Thereafter contents of the control memory are used as the address of the data memory and the data is read out to output trunk. The operation carried in one time slot is depicted in figure. The input sample is available for reading in at the beginning of the time slot and the sample is ready to be clocked in on the output stream at the end of time slot. Even if there is no time slot interchange, a sample is delayed by a minimum of one time slot in passing from the input stream to the output stream because of the storage action. In other words a time slot switch may be considered to have an inherent time delay of one time slot. In effect, the output stream is delayed by t_{TS} microseconds when compared to incoming data stream. Depending on the output time

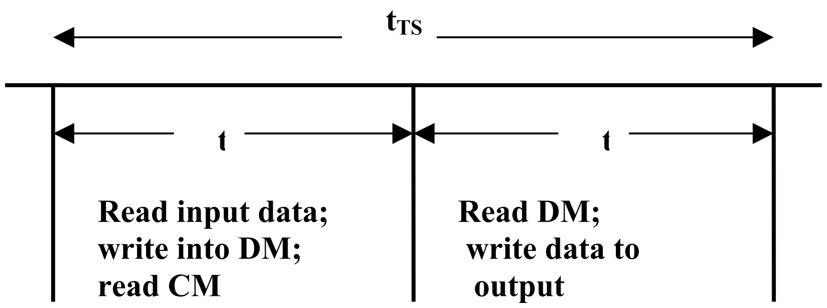


CM = control memory DM = data memory
C_{TS} = time slot clock

PRINCIPLE OF TIME SLOT INTERCHANGE

slot to which an input slot contents are switched, the sample experiences the delay in the range of t_{TS} to M t_{TS} microseconds. In the example entries, shown in the control memory of figure, the first location contains the value 1. This implies that the contents of input time slot 1 is switched to output time slot 1. The sample, in this case, experiences a delay of t_{TS} microseconds. The second location of the control memory contains the value 7 and, therefore, the input time slot 7 is switched to output time slot 2. This sample experiences a delay of $((M – 7) + 2 + 1)\ t_{TS}$ or $(M - 4)\ t_{TS}$ microseconds. Output time slot 3 carries the contents of input time slot 4 and the delay experienced by the sample is $(M – (4-3) + 1)\ t_{TS}$ or Mt_{TS}, i.e. 125 microseconds. There are two sequential memory accesses per time slot and hence the time constraint may be stated as

$t_{TS} = 2t_m$, $125 = 2Mt_m$.



t_{TS}
t
t
Read input data; write into DM; read CM
Read DM; write data to output

Operations in a time slot

where t_m is the access time of the memory modules in microseconds. When there is a two-way traffic and the network is non-folded, another set of data and control memories is used. In the second control memory, the locations 1, 7 and 4 contain the values 1, 2 and 3 respectively, corresponding to the sample entries shown in figure. When the 125µs cycle is complete, the values in the input time slots 1, 7 and 4 are interchanged with the output time slots 1,2 and 3 respectively. When the network is folded, there is only one set of data and control memories even for two way traffic. For the example shown in figure, control memory locations of 7 and 4 contain the values 2 and 3 respectively. When the 125µs cycle is complete, the values in the time slots 7 and 2 and the time slots 3 and 4 are interchanged. For a folded network, transferring the data between the same input and output slots, e.g. from input slot 1 to output slot 1, is not relevant.

Since there are no switching elements in this configuration, the cost is equal to the number of memory locations. There are M locations each in the control and in the data memory. Therefore, the cost is given by $C = 2M$ units.

4.7 SWITCH FABRICS

- Basic concepts
- Time and space switching
- Two stage switches
- Three stage switches

4.7.1 BASIC CONCEPTS

- Accessibility
- Blocking
- Complexity
- Scalability
- Reliability
- Throughput

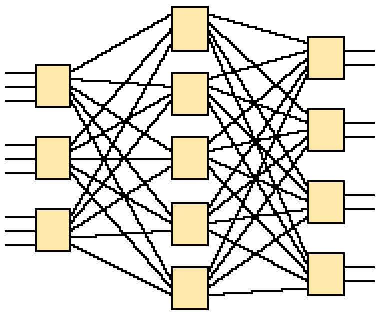
4.7.1.1 Accessibility

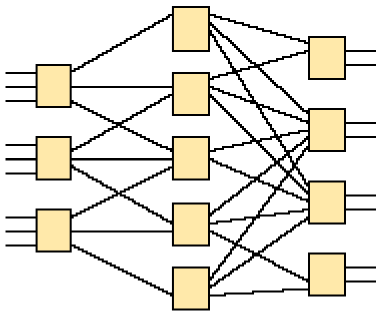
– A network has **full accessibility (=connectivity)** when each inlet can be connected to each outlet (in case there are no other I/O connections in the network)

– A network has a **limited accessibility** when the above given property does not exist

– Interconnection networks applied in today's switch fabrics usually have full accessibility

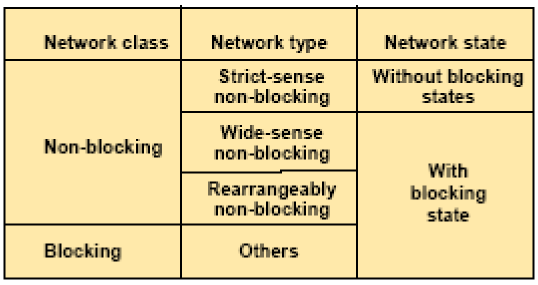
Example of full accessibility Example of limited accessibility





4.7.1.2 Blocking

Blocking is defined as failure to satisfy a connection request and it depends strongly on the combinatorial properties of the switching networks.



Network class	Network type	Network state
Non-blocking	Strict-sense non-blocking	Without blocking states
	Wide-sense non-blocking	With blocking state
	Rearrangeably non-blocking	
Blocking	Others	

• **Non-blocking** - a path between an arbitrary idle inlet and arbitrary idle outlet can always be established independent of network state at set-up time

• **Blocking** - a path between an arbitrary idle inlet and arbitrary idle outlet cannot be established owing to internal congestion due to the already established connections

• **Strict-sense non-blocking** - a path can always be set up between any idle inlet and any idle outlet without disturbing paths already set up

• **Wide-sense non-blocking** - a path can be set up between any idle inlet and any idle outlet without disturbing existing connections, provided that certain rules are followed. These rules prevent network from entering a state for which new connections cannot be made.

• **Rearrangeably non-blocking** - when establishing a path between an idle inlet and an idle outlet, paths of existing connections may have to be changed (rearranged) to set up that connection

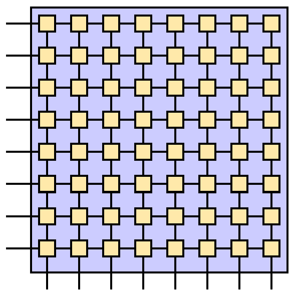
4.7.1.3 Complexity

Complexity of an interconnection network is expressed by **cost index** of switching network. Traditional definition of cost index gives the **number of cross-points in a network.** This is used to be a reasonable measure of space division switching systems.

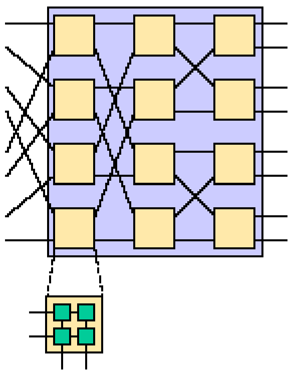
Nowadays cost index alone does not characterize cost of an interconnection network for broadband applications.

VLSIs and their integration degree has changed the way how cost of a switch fabric is formed (number of ICs, power consumption). Management and control of a switching system has a significant contribution to cost.

Cost index of an 8x8 crossbar
is 64 (cross-points)



Cost index of an 8x8 banyan
is 12x4= 48 (cross-points)



4.7.1.4 Scalability

Scalability of a switching system has become a key parameter in choosing a switch fabric architecture due to constant increase of transport links and data rates on links. Scalability describes ability of a system to evolve with increasing requirements

Issues that are usually matter of scalability are:

- number of switching nodes
- number of interconnection links between nodes
- bandwidth of interconnection links and inlets/outlets
- throughput of switch fabric
- buffering requirements
- number of inlets/outlets supported by switch fabric

Example of scalability:

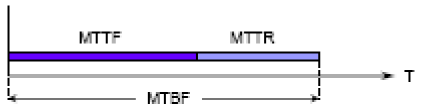
Suppose a switching equipment has room for 20 line-cards and the original design supports 10 Mbit/s interfaces (one per line card) and the throughput of switch fabrics is scalable from 500 Mbit/s to 2 Gbit/s. Original switch fabric can support new line cards that implement two 10 Mbit/s interfaces each.

Now when line interfaces are replaced with 100 Mbit/s rates(one per line-card), the switch fabric has to be updated (scaled up) to 2 Gbit/s speed. Buffering memories need to be replaced by faster (and possible larger) ones. Larger number of line cards implies at least new physical design. Increase of line rates beyond 100 Mbit/s means redesign of switch fabric.

4.7.1.5 Reliability

Reliability and fault tolerance are system measures that have an impact on all functions of a switching system. Reliability defines probability that a system does not fail within a given time interval provided that it functions correctly at the start of the interval.
Availability defines probability that a system will function at a given time instant.
Fault tolerance is the capability of a system to continue its intended function in spite of having a fault(s).
Reliability measures are:
– MTTF (Mean Time To Failure)
– MTTR (Mean Time To Repair)
– MTBF (Mean Time Between Failures)

Relation of reliability R(t) to availability F(t) is given by F(t) = 1 – R(t).
Relation of MTTF, MTTR and MTBF is shown in figure:



4.7.1.6 Throughput

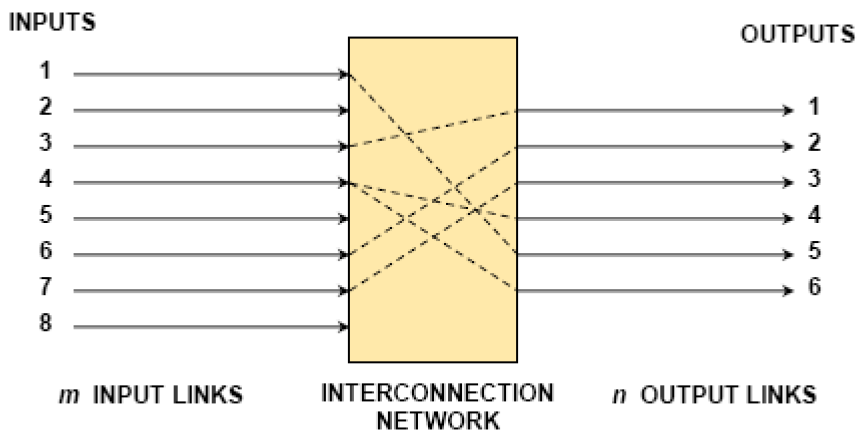
Throughput gives forwarding/switching speed/efficiency of a switch fabric. It is measured in bits/s, octets/s, cells/s, packet/s, etc. Quite often throughput is given in the range (0 ... 1.0], i.e. the obtained forwarding speed is normalized to the theoretical
maximum throughput.

4.7.2 TIME AND SPACE SWITCHING

A switched connection requires a mechanism that attaches the right information streams to each other. Switching takes place in the switching fabric, the structure of which depends on network's mode of operation, available technology and required capacity.
Communicating terminals may use different physical links and different time-slots, so there is an obvious need to switch both in time and in space domain.
Time and space switching are basic functions of a switch fabric.

4.7.2.1 SPACE DIVISION SWITCHING

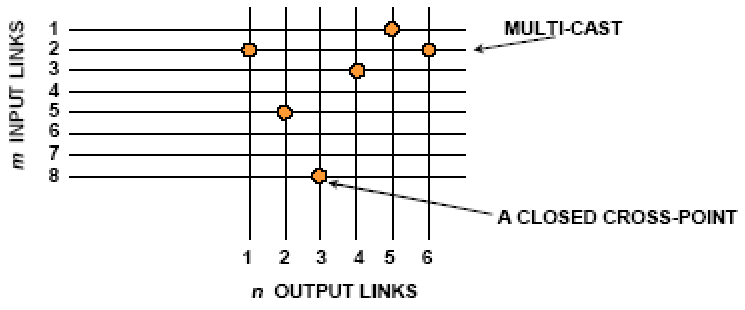
A space switch directs traffic from input links to output links. An input may set up one connection (1, 3, 6 and 7), multiple connections (4) or no connection (2, 5 and 8). In this type of switching a particular time slot is switched in the same time slot of a different channel.



Cross bar switch matrix:

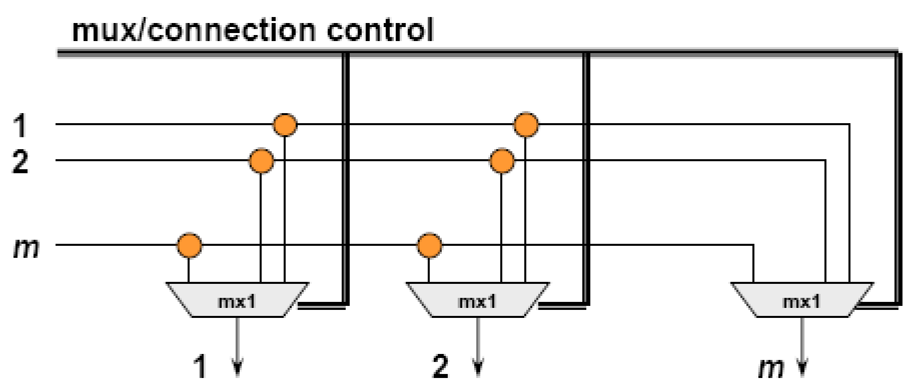
Crossbar matrix introduces the basic structure of a space switch. Information flows are controlled (switched) by opening and closing of cross-points.
For ***m*** inputs and ***n*** outputs => ***mn*** cross-points (connection points).

Only one input can be connected to an output at a time, but an input can be connected to multiple outputs (multi-cast) at a time.



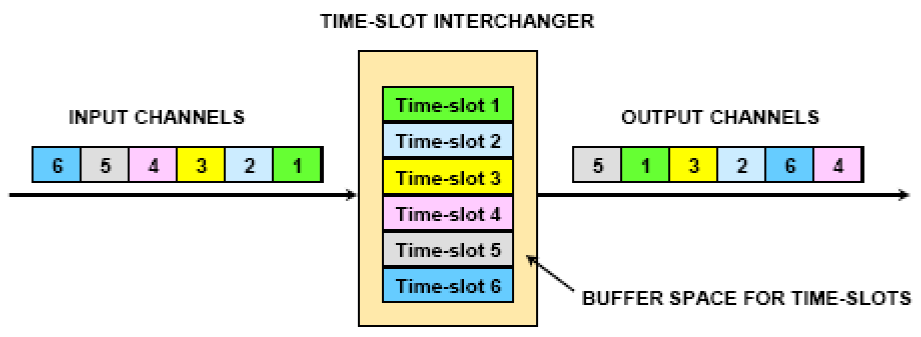
Example of space switch

*m*x1 -multiplexer used to implement a space switch. In this type of space switch every input is fed to every output multiplexer and multiplexer control signals are used to select which input signal is connected through each multiplexer.

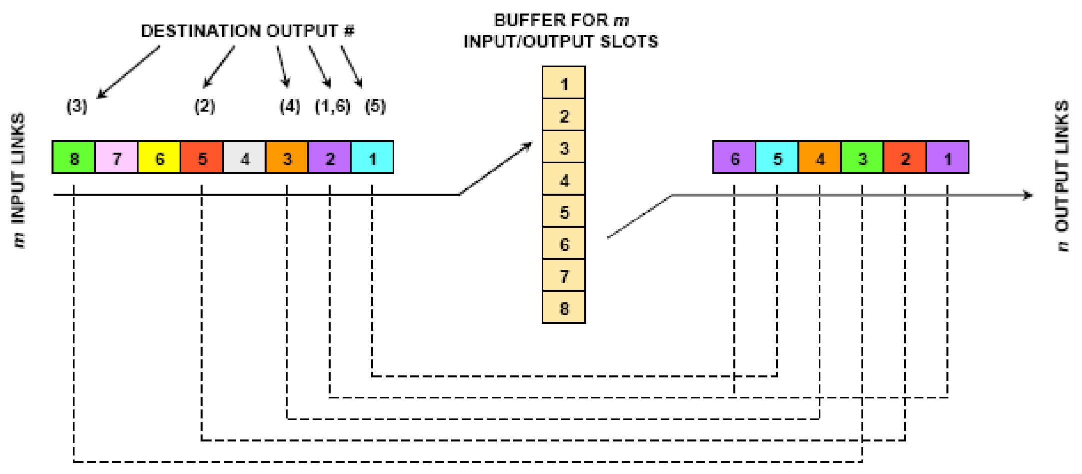


4.7.2.2 TIME DIVISION SWITCHING

Time-slot interchanger is a device, which buffers *m* incoming timeslots, e.g. 32 time-slots of an E1 frame, arranges new transmit order and transmits *n* time-slots. Time-slots are stored in buffer memory usually in the order they arrive or in the order they leave the switch. Additional control logic is needed to decide respective output order or the memory slot where an input slot is stored.

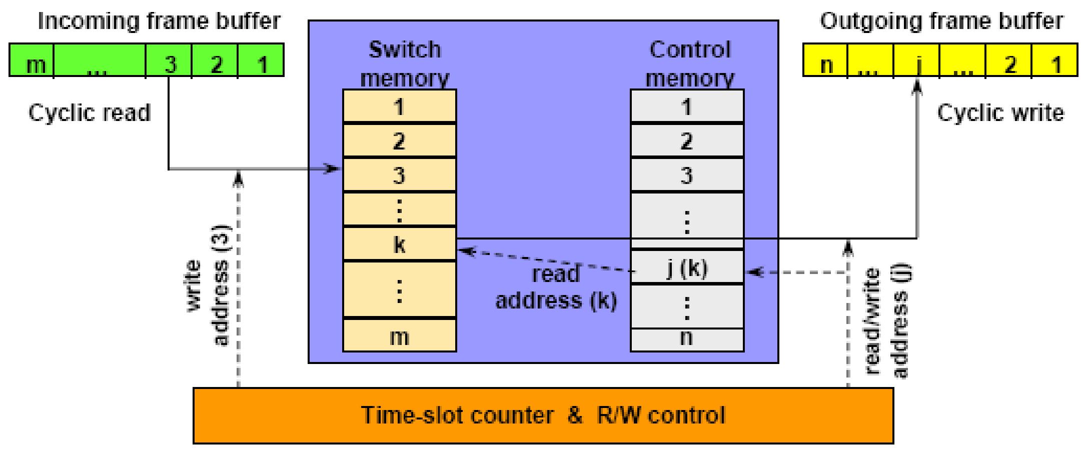


Time Slot Interchange:



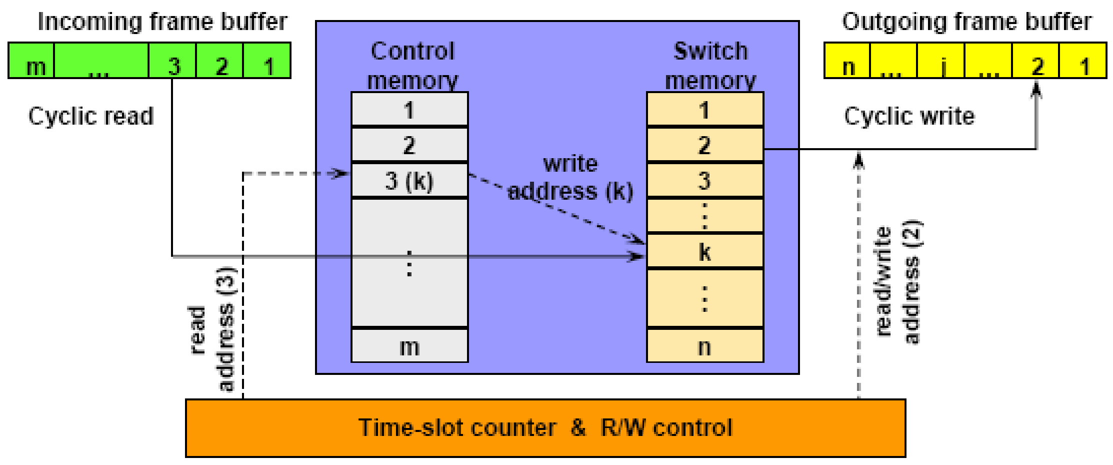
Time Switch Implementation : Example 1

Incoming time-slots are **written cyclically** into switch memory. Output logic reads cyclically control memory, which contains a pointer for each output time-slot. Pointer indicates which input time-slot to insert into each output time-slot.



Time Switch Implementation : Example 2

Incoming time-slots are **written into** switch memory by using write-addresses read from control memory. A write address points to an output slot to which the input slot is addressed. Output time-slots are read cyclically from switch memory.



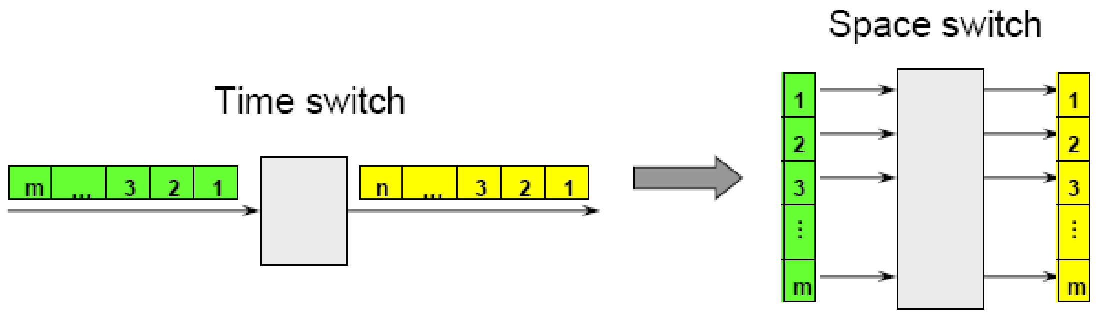
Properties of time switches

Input and output frame buffers are read and written at wire-speed, i.e. *m* R/Ws for input and *n* R/Ws for output. Interchange buffer (switch memory) serves all inputs and outputs and thus it is read and written at the aggregate speed of all inputs and outputs. Speed of an interchange buffer is a critical parameter in time switches and limits performance of a switch. Utilizing parallel to serial conversion, memory speed requirement can be cut.

Speed requirement of control memory is half of that of switch memory (in fact a little more than that to allow new control data to be updated).

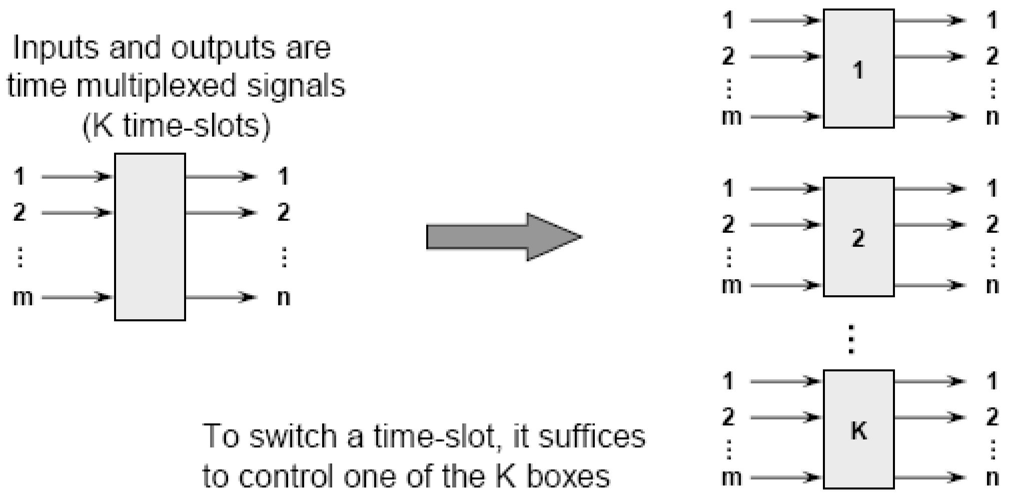
Time Space Analogy

A time switch can be logically converted into a space switch by setting time-slot buffers into vertical position => time-slots can be considered to correspond to input/output links of a space switch. But is this logical conversion fair ?



Space-Space Analogy

A space switch carrying time multiplexed input and output signals can be logically converted into a pure space switch (without cyclic control) by distributing each time-slot into its own space switch.



Inputs and outputs are time multiplexed signals (K time-slots)
1
2
m
n
1
2
K
To switch a time-slot, it suffices to control one of the K boxes

4.7.2.3 Properties of space and time switches

SPACE SWITCHES	TIME SWITCHES
• Number of cross-points (e.g. AND-gates) *m* input x *n* output = *mn* when *m*=*n* => n^2 • Output bit rate determines the speed requirement for the switch components • Both input and output lines deploy “bus” structure so fault location is difficult	• Size of switch memory (SM) and control memory (CM) grows linearly as long as memory speed is sufficient, i.e. SM + CM + input buffering + output buffering = 2 x 2 x number of time-slots • A simple and cost effective structure when memory speed is sufficient • Speed of available memory determines the maximum switching capacity

4.7.3 TWO STAGE SWITCHES

Switch fabric can be implemented as a combination of space and time switches.

This improves over all performance of switch fabric.

Two stage switches can be of following types:

• Time-Time (TT) switch

• Time-Space (TS) switch

• Space-Time (ST) switch

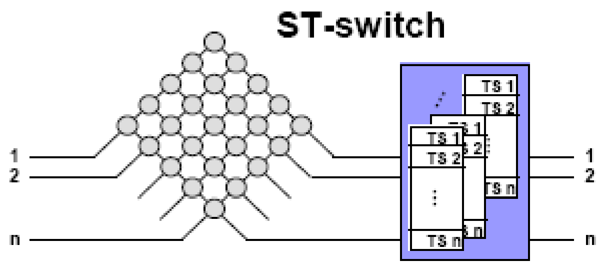
• Space-Space (SS) switch

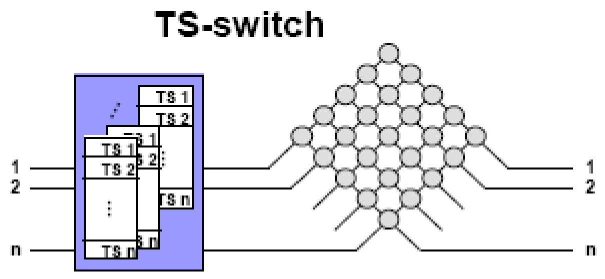
TT-switch gives no advantage compared to a single stage T-switch.

SS-switch increases blocking probability.

ST-switch gives high blocking probability (S-switch can develop blocking on an arbitrary bus, e.g. slots from two different buses attempting to flow to a common output).

TS-switch has low blocking probability, because T-switch allows rearrangement of time-slots so that S-switching can be done blocking free.



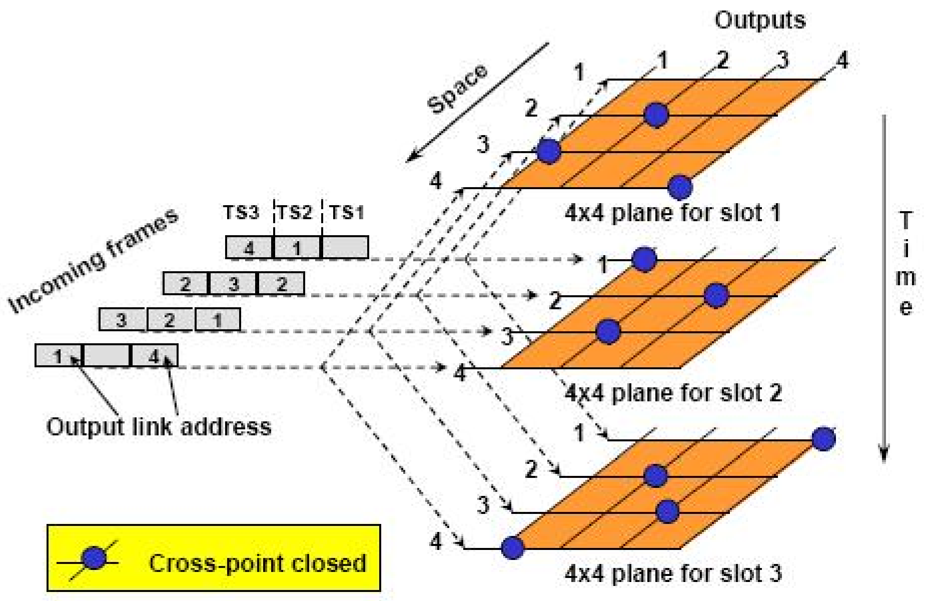


4.7.3.1 TIME MULTIPLEXED SPACE (TMS) SWITCH

In this type of switch space divided inputs and each of them carry a frame of three time-slots(in this example). Input frames on each link are synchronized to the crossbar switch.

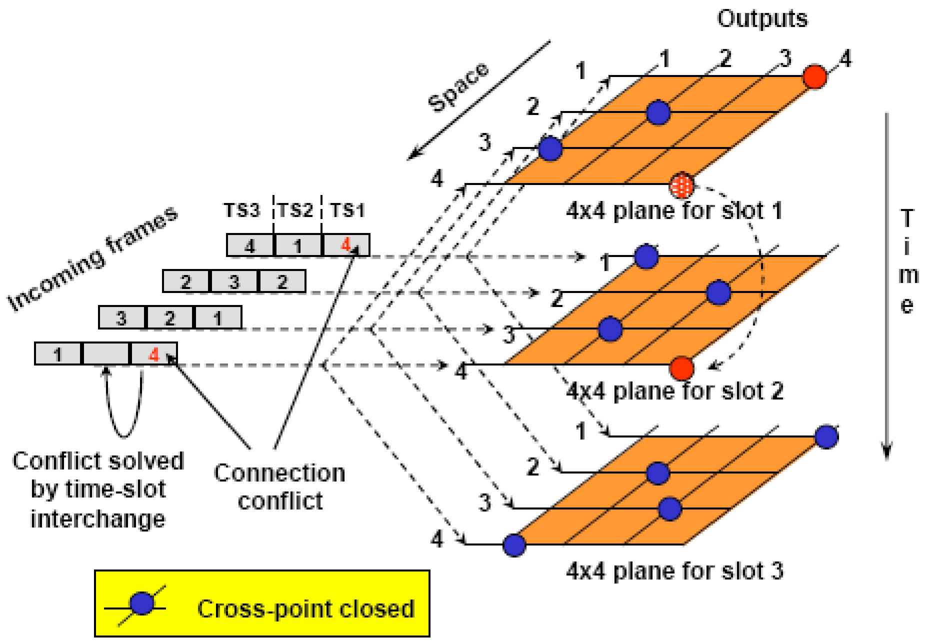
A switching plane for each time-slot is there to direct incoming slots to destined output links of the corresponding time-slot.

In the following figure space switch is shown as horizontal planes switching each time slot arriving.



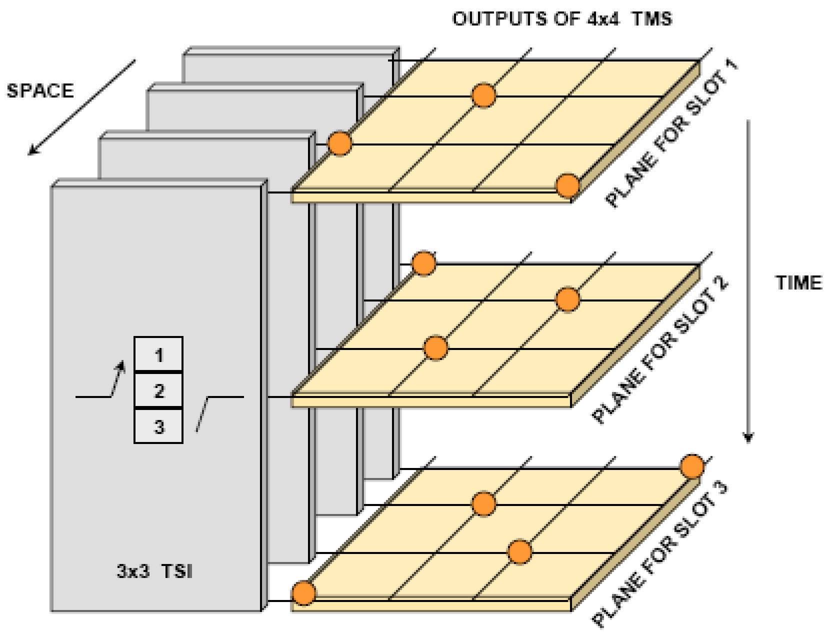
Connection conflicts in a TMS switch

If a connection request pointing to some time slot and that time slot is already occupied, then first the time slot is interchanged to resolve the conflict and then it is switched.



4.7.3.2 TS switch interconnecting TDM links

Time division switching applied prior to space switching. Incoming time-slots can always be rearranged such that output requests become conflict free for each slot of a frame, provided that the number of requests for each output is no more than the number of slots in a frame.

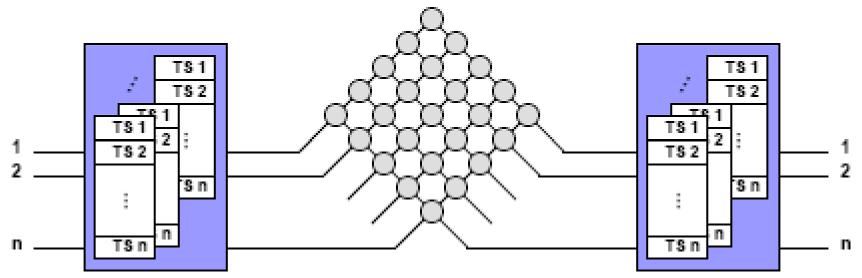


4.7.4 THREE STAGE SWITCH

Basic TS-switch sufficient for switching time-slots onto addressed outputs, but slots can appear in any order in the output frame. If a specific input slot is to carry data of a specific output slot then a time-slot interchanger is needed at each output also.

By this we have three stage switch configuration.

In this type of configuration any time-slot on any input can be connected to any time-slot on any output. And blocking probability is also minimized.
Such a 3-stage configuration is named TST-switching.



TST Switch

4.7.4.1 Three stage switch combinations

There are other combinations also possible for three stage switch. Possible three stage switch combinations are:

• Time-Time-Time (TTT) (not significant, no connection from PCM to PCM)

• Time-Time-Space (TTS) (=TS)

• Time-Space-Time (TST)
• Time-Space-Space (TSS)

• Space-Time-Time (STT) (=ST)

• Space-Time-Space (STS)

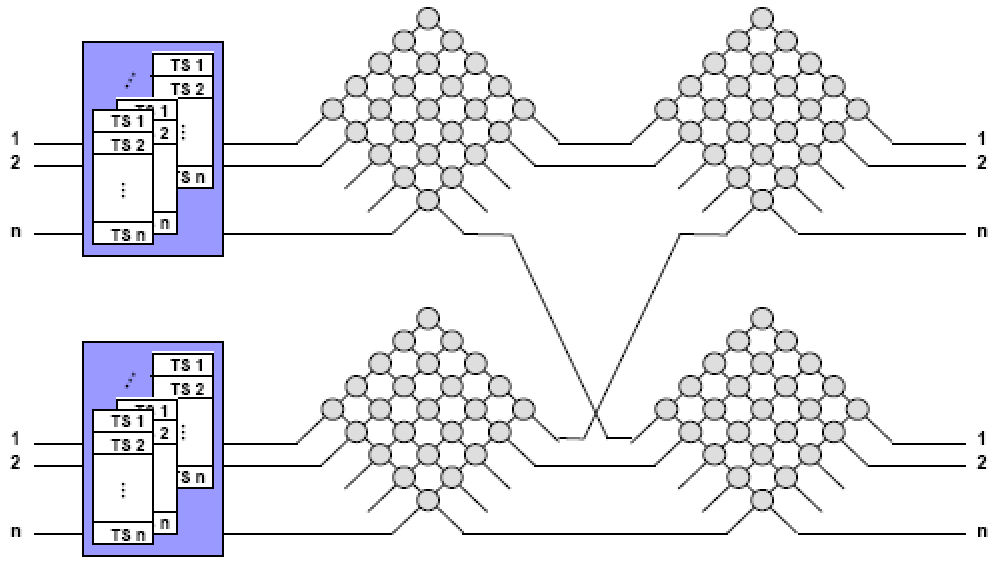
• Space-Space-Time (SST) (=ST)

• Space-Space-Space (SSS) (not significant, high probability of blocking)

• Three interesting combinations TST, TSS and STS

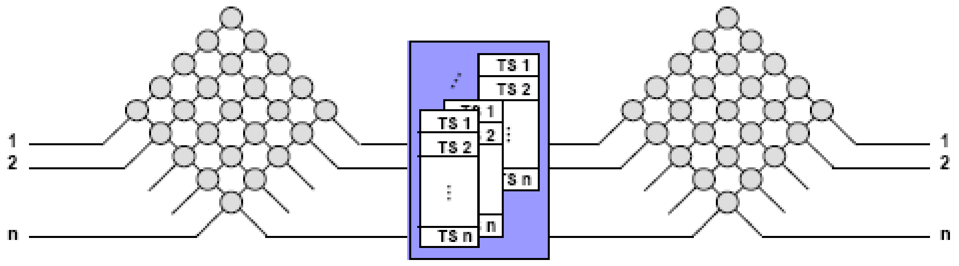
4.7.4.2 Time-Space-Space switch

Time-Space-Space switch can be applied to increase switching capacity



4.7.4.3 Space-Time-Space switch

Space-Time-Space switch has a high blocking probability (like ST-switch) - not a desired feature in public networks.



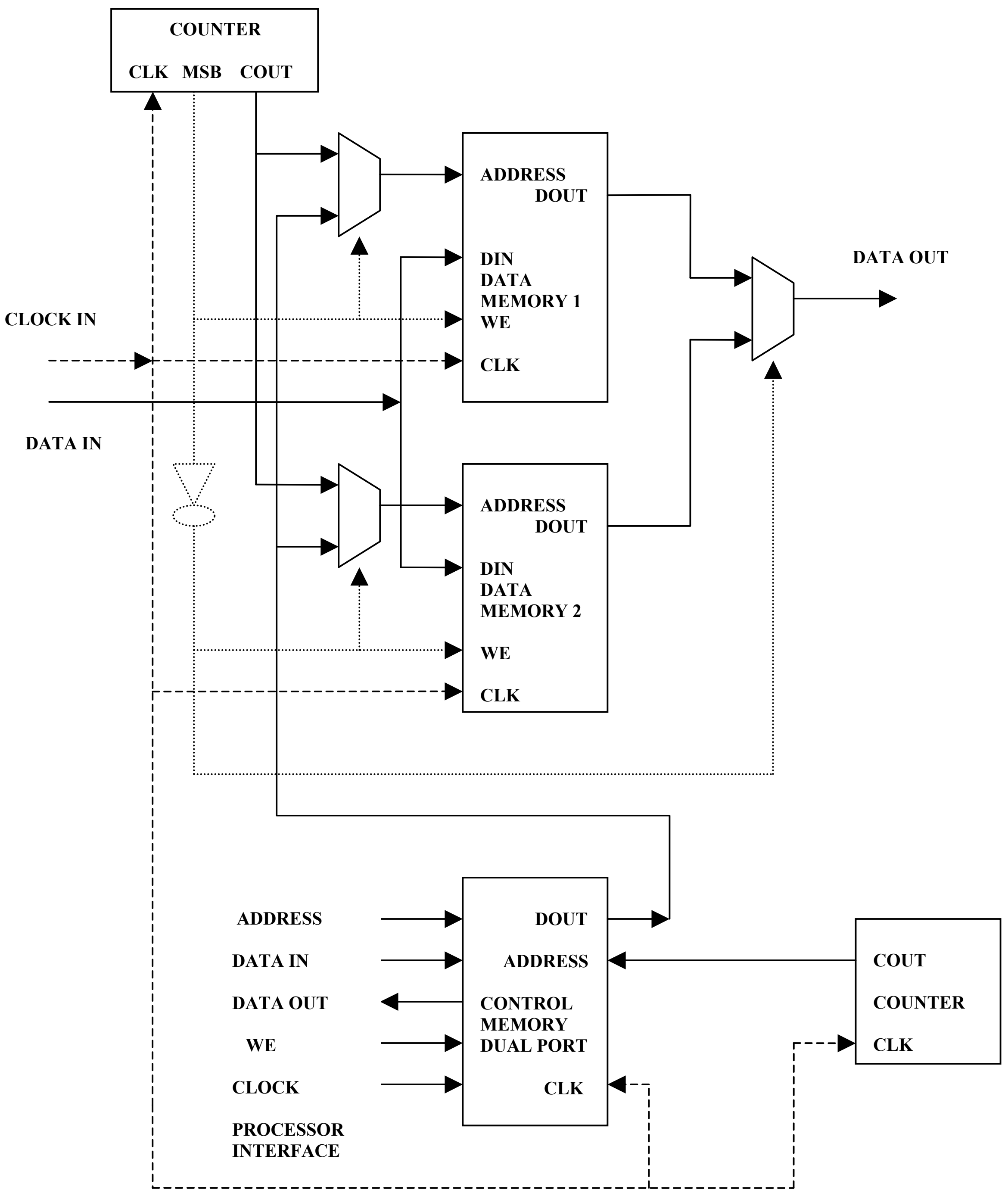
5. IMPLEMENTATION OF TIME AND SPACE SWITCH

5.1 IMPLEMENTATION OF TIME SWITCH

To achieve the functionality of time switch, the incoming data is written into data memory serially and read from data memory according to the addresses provided by control memory. For our case we consider an incoming PCM data frame of 32 bytes, which is written into data memory 1 with the same clock on which data is coming. Addresses are provided by a counter running with the same clock. So data memory capacity is of 32 X 8. Now depending upon which slot should come first locations of control memory is written by processor. Control memory is a dual port memory, so at the same time it can be written by processor from one port and read by addresses provided by a counter from other port. This counter runs with the same clock as of data rate. Control memory is of 32 X 5.

Arrangement is done so that when incoming data frame is being written into data memory 1, data for output should be read from data memory 2 and when incoming data frame is being written into data memory 2, data for output should be read from data memory 1.

5.1.2 FUNCTIONAL DIAGRAM OF TIME SWITCH



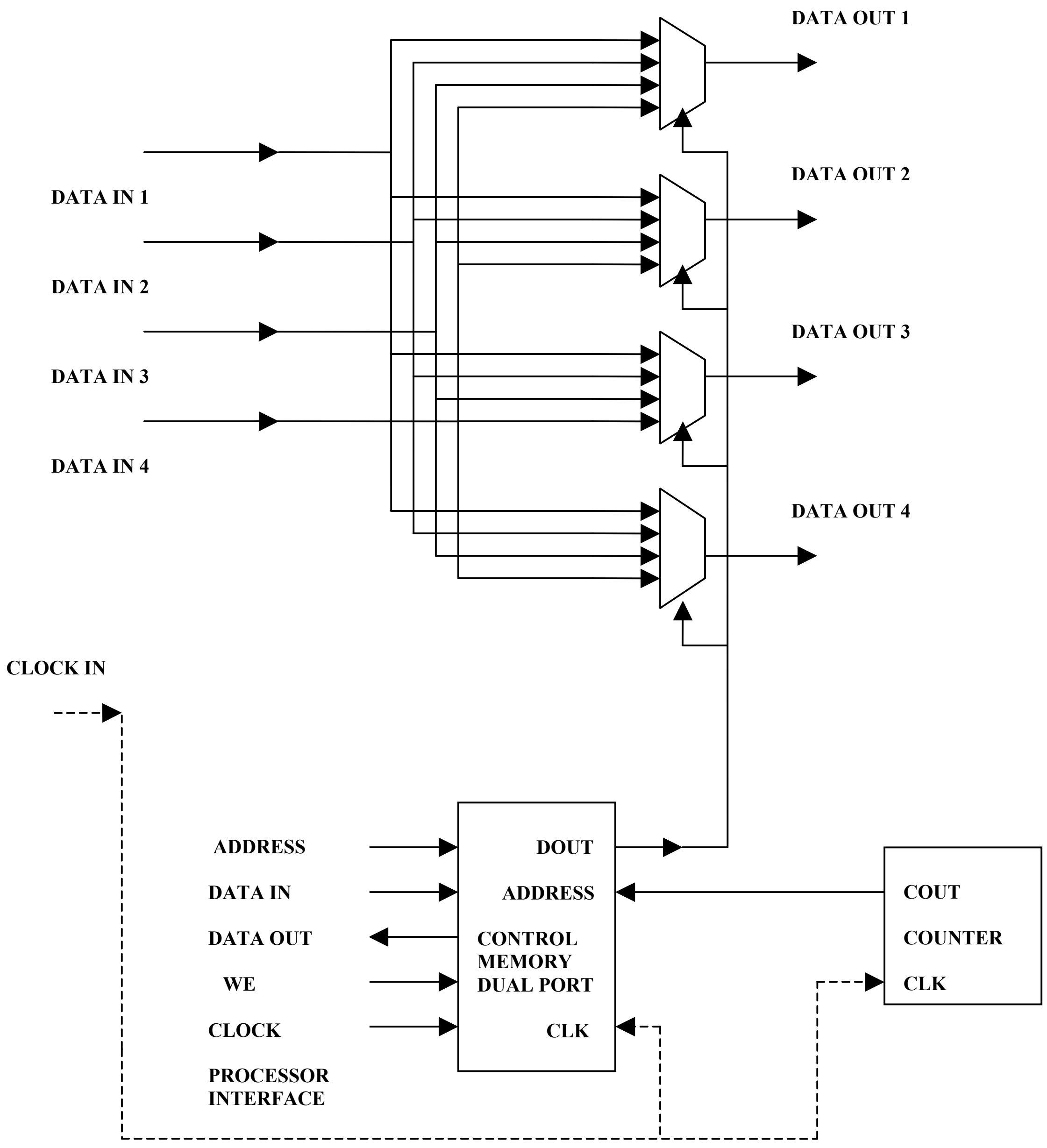
Single port block memories of 32 X 8 for Data Memories and dual port block memory of 32 X 5 for Control Memory were generated by Xilinx CoreGen. VHDL code for time switch is written by instantiating these memories.

5.2 IMPLEMENTATION OF SPACE SWITCH

To achieve the functionality of space switch, the incoming data is given to the input of multiplexers. We have chosen four multiplexers for a 4 X 4 space switch.

Control signals for the multiplexers are provided by control memory. There are four buses going to each multiplexers so we need two control signals for each multiplexer. So, total eight control signals are needed per time slot. Control memory capacity is of 32 X 8. Because there are 32 slots and eight control signals per slot. Control memory is a dual port memory, so at the same time it can be written by processor from one port and read by addresses provided by a counter from another port. This counter runs with the same clock as of data rate. Thus we can change the space switch configuration on per time slot basis, this type of space switch is called Time Multiplexed Space Switch.

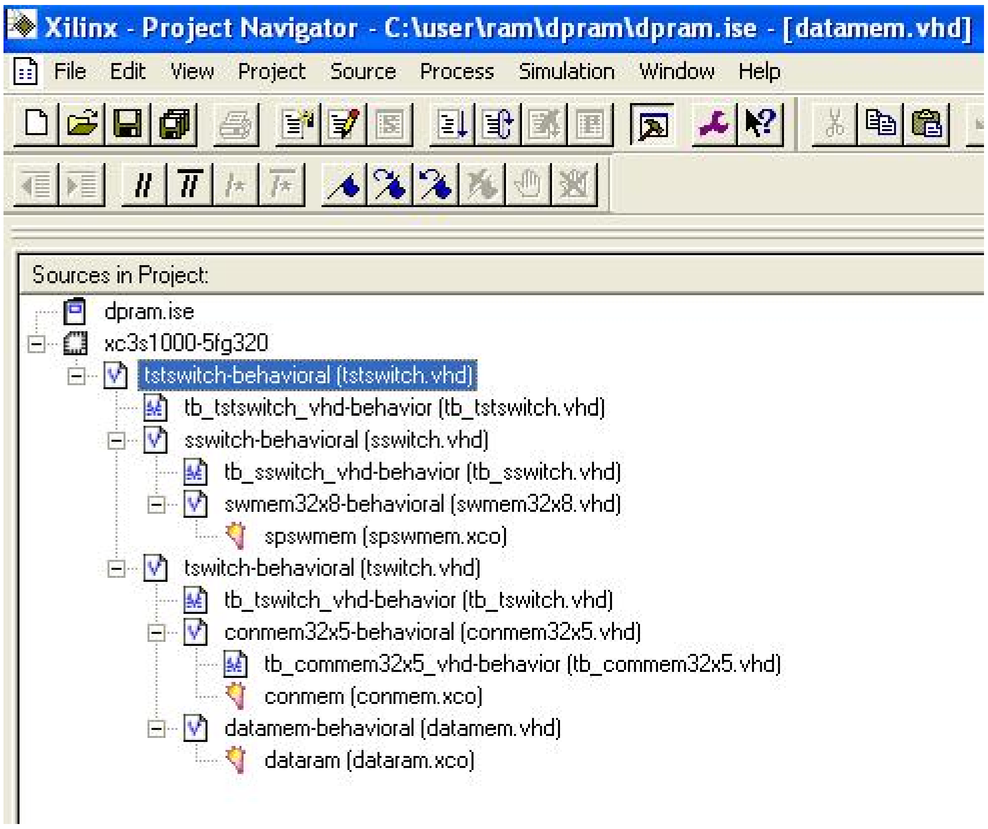
5.2.2 FUNCTIONAL DIAGRAM OF SPACE SWITCH



Dual port block memory of 32 X 8 for Control Memory was generated by Xilinx CoreGen software. VHDL code for space switch switch is written by instantiating this memory.

5.3 VHDL CODES

5.3.1 PROJECT HEIRARCHY


Xilinx - Project Navigator - C:\user\ram\dpram\dpram.ise - [datamem.vhd]
File
Edit
View
Project
Source
Process
Simulation
Window
Help

```
Sources in Project:
dpram.ise
xc3s1000-5fg320
  tstswitch-behavioral (tstswitch.vhd)
    tb_tstswitch_vhd-behavior (tb_tstswitch.vhd)
    sswitch-behavioral (sswitch.vhd)
      tb_sswitch_vhd-behavior (tb_sswitch.vhd)
      swmem32x8-behavioral (swmem32x8.vhd)
        spswmem (spswmem.xco)
    tswitch-behavioral (tswitch.vhd)
      tb_tswitch_vhd-behavior (tb_tswitch.vhd)
      conmem32x5-behavioral (conmem32x5.vhd)
        tb_commem32x5_vhd-behavior (tb_commem32x5.vhd)
        conmem (conmem.xco)
      datamem-behavioral (datamem.vhd)
        dataram (dataram.xco)
```

- ‘tswitch’ is entity of time switch. ‘tb_tswitch’ is test bench of ‘tswitch’. It uses xilinx coregen component ‘conmem’ and ‘dataram’ instantiated by ‘conmem32x5’ and ‘datamem’.
- ‘sswitch’ is entity of space switch. ‘tb_sswitch’ is test bench of ‘sswitch’. It uses xilinx coregen component ‘spswmem’ instantiated by ‘swmem32x8’.
- ‘tstswitch’ is 4x4 time-space-time switch instantiating ‘tswitch’ and ‘sswitch’. ‘tb_tstswitch’ is its test bench.

5.3.2 VHDL CODE OF TST SWITCH

```
--------------------------------------------------
-- Company:
-- Engineer:      Praveen
-- Create Date:    13:45:49 06/21/06
-- Design Name:
-- Module Name:    tstswitch - Behavioral
-- Project Name:
-- Target Device:
-- Tool versions:
-- Description:
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--------------------------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity tstswitch is
    Port ( ip1 : in std_logic_vector(7 downto 0);
           ip2 : in std_logic_vector(7 downto 0);
           ip3 : in std_logic_vector(7 downto 0);
           ip4 : in std_logic_vector(7 downto 0);
           op1 : out std_logic_vector(7 downto 0);
           op2 : out std_logic_vector(7 downto 0);
           op3 : out std_logic_vector(7 downto 0);
           op4 : out std_logic_vector(7 downto 0);
           reset : in std_logic;
           clkin : in std_logic;
           proc_addr : in std_logic_vector(8 downto 0);
           proc_datain : in std_logic_vector(7 downto 0);
           proc_dataout : out std_logic_vector(7 downto 0);
           proc_clk : in std_logic;
           proc_rwb : in std_logic;
           proc_csb : in std_logic);
end tstswitch;

architecture Behavioral of tstswitch is

component tswitch is
    Port ( proc_addr : in std_logic_vector(4 downto 0);
           proc_data_in : in std_logic_vector(7 downto 0);
           proc_data_out : out std_logic_vector(7 downto 0);
           csb : in std_logic;
           rwb : in std_logic;
           proc_clk : in std_logic;
           reset : in std_logic;
           data_in : in std_logic_vector(7 downto 0);
           data_out : out std_logic_vector(7 downto 0);
                    count_out : out std_logic_vector(4 downto 0);
                    con_mem_data :  out std_logic_vector(4 downto 0);
           clk : in std_logic);
end component;

component sswitch is
    Port ( addr_sw : in std_logic_vector(4 downto 0);
           din_sw : in std_logic_vector(7 downto 0);
           dout_sw : out std_logic_vector(7 downto 0);
           clk_sw : in std_logic;
           rwb_sw : in std_logic;
           csb_sw : in std_logic;
           ip1 : in std_logic_vector(7 downto 0);
           ip2 : in std_logic_vector(7 downto 0);
           ip3 : in std_logic_vector(7 downto 0);
           ip4 : in std_logic_vector(7 downto 0);
           op1 : out std_logic_vector(7 downto 0);
           op2 : out std_logic_vector(7 downto 0);
           op3 : out std_logic_vector(7 downto 0);
           op4 : out std_logic_vector(7 downto 0);
           count_out : out std_logic_vector(4 downto 0);
                    clkin : in std_logic;
                    reset : in std_logic);
```

```
end component;

signal csb1,csb2,csb3,csb4,csb5 : std_logic;
signal csb6,csb7,csb8,csb9      : std_logic;
signal iip1,iip2,iip3,iip4 : std_logic_vector(7 downto 0);
signal oop1,oop2,oop3,oop4 : std_logic_vector(7 downto 0);
signal count_out1,count_out2,count_out3:std_logic_vector(4 downto 0);
signal count_out4,count_out5,count_out6:std_logic_vector(4 downto 0);
signal count_out7,count_out8,count_out9:std_logic_vector(4 downto 0);
signal con_mem_data1,con_mem_data2 :std_logic_vector(4 downto 0);
signal con_mem_data3,con_mem_data4 :std_logic_vector(4 downto 0);
signal con_mem_data5,con_mem_data6 :std_logic_vector(4 downto 0);
signal con_mem_data7,con_mem_data8 :std_logic_vector(4 downto 0);
signal chipsel :std_logic_vector(3 downto 0);

begin  --of architecture

 chipsel(3 downto 0) <= proc_addr(8 downto 5);

process(chipsel) --generating enble signals for different memories
begin
case chipsel is
when "0000" => csb1<='0';csb2<='1';csb3<='1';csb4<='1';csb5<='1';
               csb6<='1';csb7<='1';csb8<='1';csb9<='1';
when "0001" => csb1<='1';csb2<='0';csb3<='1';csb4<='1';csb5<='1';
               csb6<='1';csb7<='1';csb8<='1';csb9<='1';
when "0010" => csb1<='1';csb2<='1';csb3<='0';csb4<='1';csb5<='1';
               csb6<='1';csb7<='1';csb8<='1';csb9<='1';
when "0011" => csb1<='1';csb2<='1';csb3<='1';csb4<='0';csb5<='1';
               csb6<='1';csb7<='1';csb8<='1';csb9<='1';
when "0100" => csb1<='1';csb2<='1';csb3<='1';csb4<='1';csb5<='0';
               csb6<='1';csb7<='1';csb8<='1';csb9<='1';
when "0101" => csb1<='1';csb2<='1';csb3<='1';csb4<='1';csb5<='1';
               csb6<='0';csb7<='1';csb8<='1';csb9<='1';
when "0110" => csb1<='1';csb2<='1';csb3<='1';csb4<='1';csb5<='1';
               csb6<='1';csb7<='0';csb8<='1';csb9<='1';
when "0111" => csb1<='1';csb2<='1';csb3<='1';csb4<='1';csb5<='1';
               csb6<='1';csb7<='1';csb8<='0';csb9<='1';
when "1000" => csb1<='1';csb2<='1';csb3<='1';csb4<='1';csb5<='1';
               csb6<='1';csb7<='1';csb8<='1';csb9<='0';
when others => csb1<='1';csb2<='1';csb3<='1';csb4<='1';csb5<='1';
               csb6<='1';csb7<='1';csb8<='1';csb9<='1';
end case;
end process;

-- instantiating eight time switches and one space switch
-- one time switch each for four incoming and four outgoing buses

ts1 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb1 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => ip1 ,
            data_out => iip1 ,
            count_out => count_out1,
            con_mem_data => con_mem_data1,
            clk => clkin      );

ts2 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb2 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => ip2 ,
            data_out => iip2 ,
            count_out => count_out2,
            con_mem_data => con_mem_data2,
            clk => clkin      );

ts3 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb3 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => ip3 ,
            data_out => iip3 ,
            count_out => count_out3,
            con_mem_data => con_mem_data3,
            clk => clkin      );

ts4 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb4 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => ip4 ,
            data_out => iip4 ,
            count_out => count_out4,
            con_mem_data => con_mem_data4,
            clk => clkin      );

ts5 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb5 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => oop1 ,
            data_out => op1 ,
            count_out => count_out5,
            con_mem_data => con_mem_data5,
            clk => clkin      );

ts6 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb6 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => oop2 ,
            data_out => op2 ,
            count_out => count_out6,
            con_mem_data => con_mem_data6,
            clk => clkin      );

ts7 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb7 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => oop3 ,
            data_out => op3 ,
            count_out => count_out7,
            con_mem_data => con_mem_data7,
            clk => clkin      );

ts8 : tswitch PORT MAP(
            proc_addr => proc_addr(4 downto 0),
            proc_data_in => proc_datain,
            proc_data_out => proc_dataout,
            csb => csb8 ,
            rwb => proc_rwb,
            proc_clk => proc_clk,
            reset => reset,
            data_in => oop4 ,
            data_out => op4 ,
            count_out => count_out8,
            con_mem_data => con_mem_data8,
            clk => clkin      );

ss1 : sswitch PORT MAP(
            addr_sw => proc_addr(4 downto 0),
            din_sw => proc_datain,
            dout_sw => proc_dataout,
            clk_sw => proc_clk,
            rwb_sw => proc_rwb,
            csb_sw => csb9 ,
            ip1 => iip1,
            ip2 => iip2,
            ip3 => iip3,
            ip4 => iip4,
            op1 => oop1,
            op2 => oop2,
            op3 => oop3,
            op4 => oop4,
            count_out => count_out9,
            clkin => clkin,
            reset => reset );

end Behavioral;

---------------------------------------------
```

5.3.3 VHDL CODE OF TIME SWITCH

```
------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
```

```
--library UNISIM;
--use UNISIM.VComponents.all;

entity tswitch is
    Port ( proc_addr : in std_logic_vector(4 downto 0);
           proc_data_in : in std_logic_vector(7 downto 0);
           proc_data_out : out std_logic_vector(7 downto 0);
           csb : in std_logic;
           rwb : in std_logic;
           proc_clk : in std_logic;
           reset : in std_logic;
           data_in : in std_logic_vector(7 downto 0);
           data_out : out std_logic_vector(7 downto 0);
                    count_out : out std_logic_vector(4 downto 0);
                    con_mem_data :  out std_logic_vector(4 downto 0);
           clk : in std_logic);
end tswitch;

architecture Behavioral of tswitch is

component datamem is
    Port ( datain : in std_logic_vector(7 downto 0);
           clk : in std_logic;
           dataout : out std_logic_vector(7 downto 0);
           con_datain : in std_logic_vector(4 downto 0);
           reset : in std_logic);
end component;

component conmem32x5 is
    Port ( proc_addr : in std_logic_vector(4 downto 0);
           proc_data_in : in std_logic_vector(7 downto 0);
           proc_clk : in std_logic;
           csb : in std_logic;
           rwb : in std_logic;
           proc_data_out : out std_logic_vector(7 downto 0);
           reset : in std_logic;
           clk : in std_logic;
                    count_out : out std_logic_vector(4 downto 0);
           data_out : out std_logic_vector(4 downto 0));
end component;

signal condata : std_logic_vector(4 downto 0);

begin

 con_mem_data <= condata ;

 -- instantiating data memory
 module1 : datamem
        port map (
           datain => data_in,
           clk    => clk,
           dataout => data_out,
           con_datain => condata,
           reset =>  reset);

 -- instantiating control memory
 module2 : conmem32x5
      port map (
                    proc_addr => proc_addr,
           proc_data_in => proc_data_in,
           proc_clk => proc_clk,
           csb => csb ,
           rwb => rwb ,
           proc_data_out => proc_data_out,
           reset => reset,
           clk => clk ,
                    count_out => count_out ,
           data_out => condata);

end Behavioral;
```

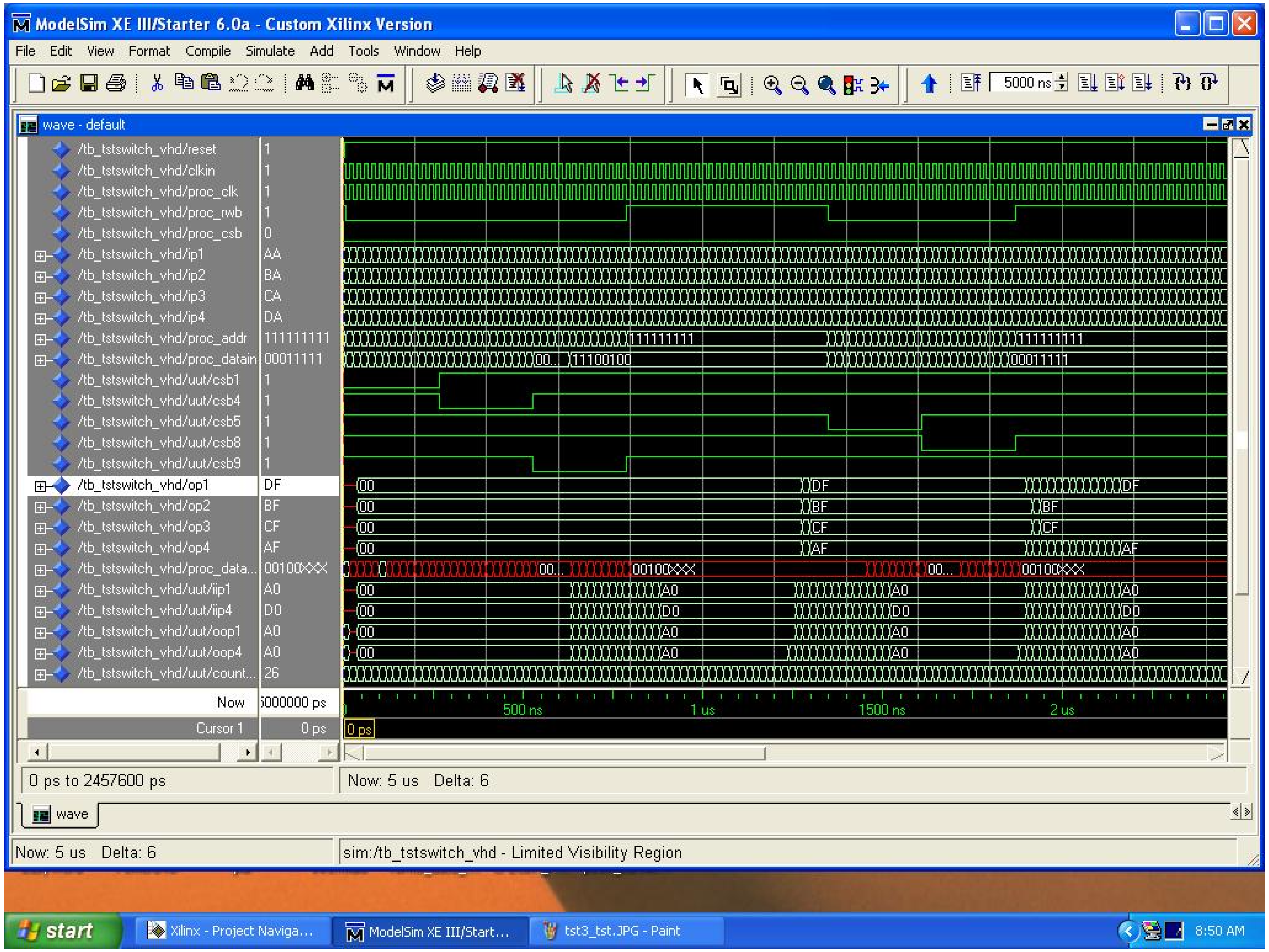
5.3.4 VHDL CODE OF SPACE SWITCH

```
------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sswitch is
    Port ( addr_sw : in std_logic_vector(4 downto 0);
           din_sw : in std_logic_vector(7 downto 0);
           dout_sw : out std_logic_vector(7 downto 0);
           clk_sw : in std_logic;
           rwb_sw : in std_logic;
           csb_sw : in std_logic;
           ip1 : in std_logic_vector(7 downto 0);
           ip2 : in std_logic_vector(7 downto 0);
           ip3 : in std_logic_vector(7 downto 0);
           ip4 : in std_logic_vector(7 downto 0);
           op1 : out std_logic_vector(7 downto 0);
           op2 : out std_logic_vector(7 downto 0);
           op3 : out std_logic_vector(7 downto 0);
           op4 : out std_logic_vector(7 downto 0);
           count_out : out std_logic_vector(4 downto 0);
                    clkin : in std_logic;
                    reset : in std_logic);
end sswitch;

architecture Behavioral of sswitch is

component swmem32x8 is
    Port ( addra_sw : in std_logic_vector(4 downto 0);
           dina_sw : in std_logic_vector(7 downto 0);
           douta_sw : out std_logic_vector(7 downto 0);
           clka_sw : in std_logic;
           rwba_sw : in std_logic;
           csba_sw : in std_logic;
                    addrb_sw : in std_logic_vector(4 downto 0);
                    doutb_sw : out std_logic_vector(7 downto 0);
           clkb_sw : in std_logic);

end component;

signal   addra_sw :  std_logic_vector(4 downto 0);
signal   dina_sw :  std_logic_vector(7 downto 0);
signal   douta_sw : std_logic_vector(7 downto 0);
signal   clka_sw :  std_logic;
signal   rwba_sw :  std_logic;
signal   csba_sw :  std_logic;
signal   count :  std_logic_vector(4 downto 0);
signal   ctrl  : std_logic_vector(7 downto 0);
signal   ctrl1,ctrl2,ctrl3,ctrl4:std_logic_vector(1 downto 0);

begin
 -- instantiating space switch control memory
swmemory: swmem32x8 port map
      (       addra_sw => addr_sw ,
           dina_sw  => din_sw ,
           douta_sw => dout_sw  ,
           clka_sw  => clk_sw  ,
           rwba_sw  => rwb_sw ,
           csba_sw  => csb_sw ,
                    addrb_sw => count ,
           doutb_sw => ctrl  ,
           clkb_sw  => clkin  );


process(reset,clkin)
begin
 if reset ='0' then count <= "00000";
 elsif clkin'event and clkin='1' then
 count <= count + 1;
 end if;
end process;

process(ctrl1,ip1,ip2,ip3,ip4)
begin    -- implementing multiplexers
case ctrl1 is
when "00" => op1 <= ip1 ;
when "01" => op1 <= ip2 ;
when "10" => op1 <= ip3 ;
when "11" => op1 <= ip4 ;
when others => null ;
end case;
end process;
process(ctrl2,ip1,ip2,ip3,ip4)
begin
case ctrl2 is
when "00" => op2 <= ip1 ;
when "01" => op2 <= ip2 ;
when "10" => op2 <= ip3 ;
when "11" => op2 <= ip4 ;
when others => null ;
end case;
end process;
process(ctrl3,ip1,ip2,ip3,ip4)
begin
case ctrl3 is
when "00" => op3 <= ip1 ;
when "01" => op3 <= ip2 ;
when "10" => op3 <= ip3 ;
when "11" => op3 <= ip4 ;
when others => null ;
end case;
end process;
process(ctrl4,ip1,ip2,ip3,ip4)
begin
case ctrl4 is
when "00" => op4 <= ip1 ;
when "01" => op4 <= ip2 ;
when "10" => op4 <= ip3 ;
when "11" => op4 <= ip4 ;
when others => null ;
end case;
end process;
```

```
ctrl1 <= ctrl(1 downto 0);
ctrl2 <= ctrl(3 downto 2);
ctrl3 <= ctrl(5 downto 4);
ctrl4 <= ctrl(7 downto 6);
count_out <= count;

end Behavioral;
```

5.4 SIMULATION RESULTS

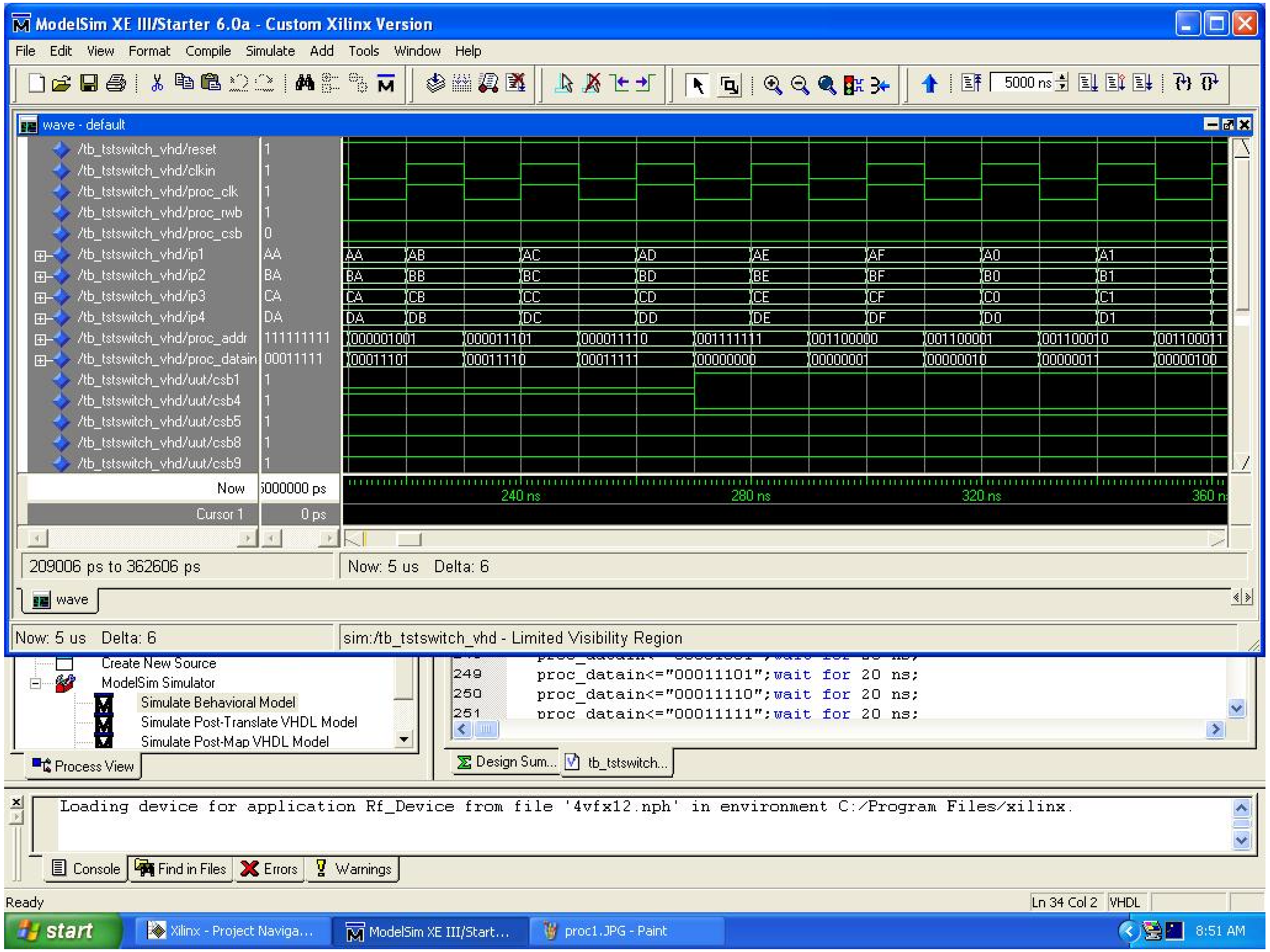
- TIME SWITCHING
- SPACE SWITCHING
- TIME-SPACE SWITCHING
- TIME-SPACE-TIME SWITCHING 1
- TIME-SPACE-TIME SWITCHING 1
- TIME-SPACE-TIME SWITCHING 1
- PROCESSOR ACCES 1
- PROCESSOR ACCES 2


ModelSim XE III/Starter 6.0a - Custom Xilinx Version

wave - default

```
/tb_tstswitch_vhd/reset            1
/tb_tstswitch_vhd/clkin            1
/tb_tstswitch_vhd/proc_clk         1
/tb_tstswitch_vhd/proc_rwb         1
/tb_tstswitch_vhd/proc_csb         0
/tb_tstswitch_vhd/ip1              AA
/tb_tstswitch_vhd/ip2              BA
/tb_tstswitch_vhd/ip3              CA
/tb_tstswitch_vhd/ip4              DA
/tb_tstswitch_vhd/proc_addr        111111111
/tb_tstswitch_vhd/proc_datain      00011111
/tb_tstswitch_vhd/uut/csb1         1
/tb_tstswitch_vhd/uut/csb4         1
/tb_tstswitch_vhd/uut/csb5         1
/tb_tstswitch_vhd/uut/csb8         1
/tb_tstswitch_vhd/uut/csb9         1
/tb_tstswitch_vhd/op1              DF
/tb_tstswitch_vhd/op2              BF
/tb_tstswitch_vhd/op3              CF
/tb_tstswitch_vhd/op4              AF
/tb_tstswitch_vhd/proc_data...     00100XXX
/tb_tstswitch_vhd/uut/iip1         A0
/tb_tstswitch_vhd/uut/iip4         D0
/tb_tstswitch_vhd/uut/oop1         A0
/tb_tstswitch_vhd/uut/oop4         A0
/tb_tstswitch_vhd/uut/count...     26
Now                                5000000 ps
Cursor 1                           0 ps
```

```
0 ps to 2457600 ps        Now: 5 us   Delta: 6
Now: 5 us   Delta: 6      sim:/tb_tstswitch_vhd - Limited Visibility Region
```



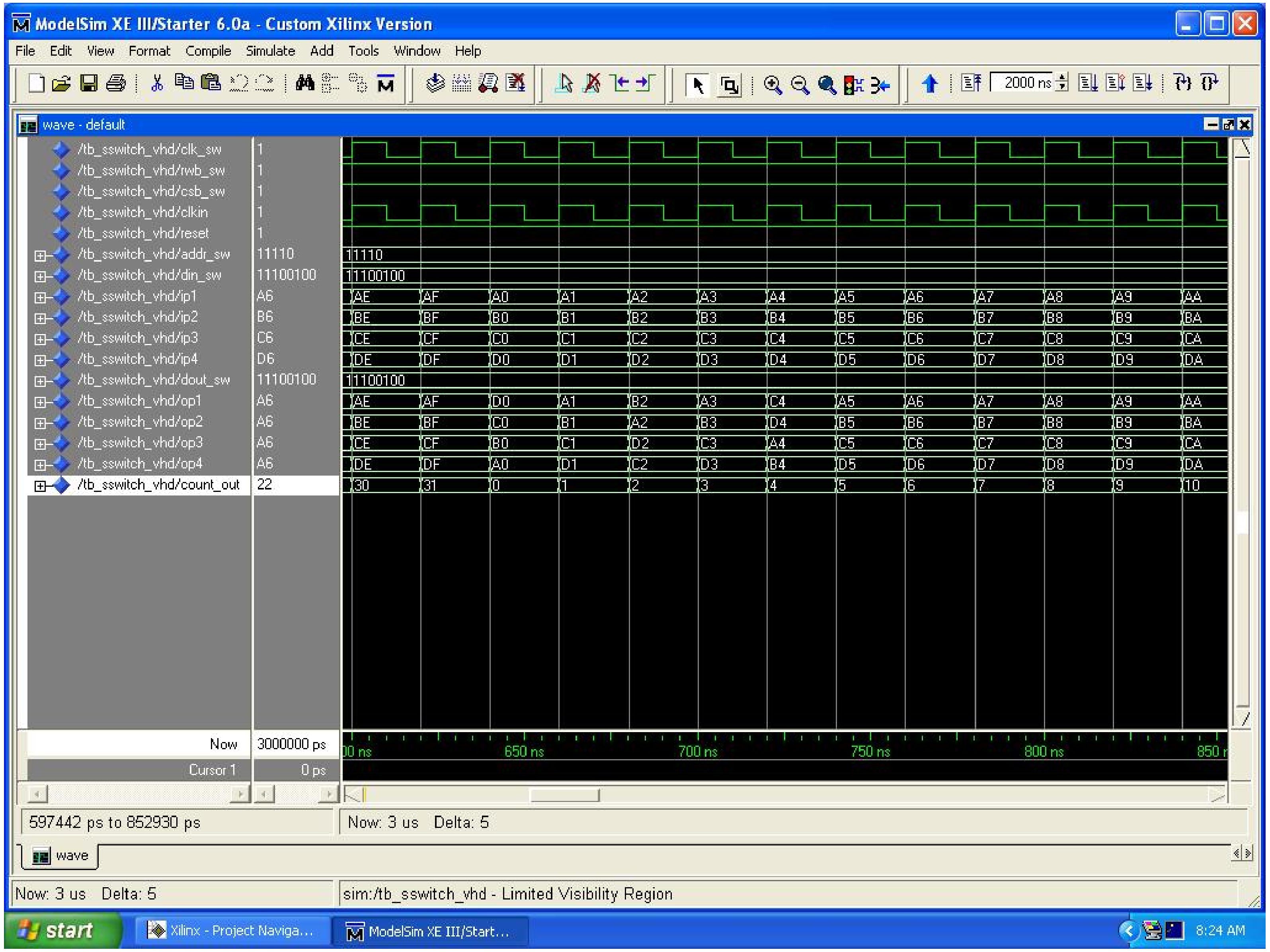
```
/tb_tstswitch_vhd/reset          1
/tb_tstswitch_vhd/clkin          1
/tb_tstswitch_vhd/proc_clk       1
/tb_tstswitch_vhd/proc_rwb       1
/tb_tstswitch_vhd/proc_csb       0
/tb_tstswitch_vhd/ip1            AA         AA  AB  AC  AD  AE  AF  A0  A1
/tb_tstswitch_vhd/ip2            BA         BA  BB  BC  BD  BE  BF  B0  B1
/tb_tstswitch_vhd/ip3            CA         CA  CB  CC  CD  CE  CF  C0  C1
/tb_tstswitch_vhd/ip4            DA         DA  DB  DC  DD  DE  DF  D0  D1
/tb_tstswitch_vhd/proc_addr      111111111  000001001 000011101 000011110 001111111 001100000 001100001 001100010 001100011
/tb_tstswitch_vhd/proc_datain    00011111   00011101 00011110 00011111 00000000 00000001 00000010 00000011 00000100
/tb_tstswitch_vhd/uut/csb1       1
/tb_tstswitch_vhd/uut/csb4       1
/tb_tstswitch_vhd/uut/csb5       1
/tb_tstswitch_vhd/uut/csb8       1
/tb_tstswitch_vhd/uut/csb9       1
Now      5000000 ps      240 ns      280 ns      320 ns      360 ns
Cursor 1 0 ps

209006 ps to 362606 ps     Now: 5 us  Delta: 6

Now: 5 us  Delta: 6     sim:/tb_tstswitch_vhd - Limited Visibility Region
```

```
Create New Source
ModelSim Simulator
    Simulate Behavioral Model
    Simulate Post-Translate VHDL Model
    Simulate Post-Map VHDL Model
```

```
proc_datain<="00011101";wait for 20 ns;
proc_datain<="00011110";wait for 20 ns;
proc_datain<="00011111";wait for 20 ns;
```

```
Loading device for application Rf_Device from file '4vfx12.nph' in environment C:/Program Files/xilinx.
```

```
Ready                                   Ln 34 Col 2   VHDL
```



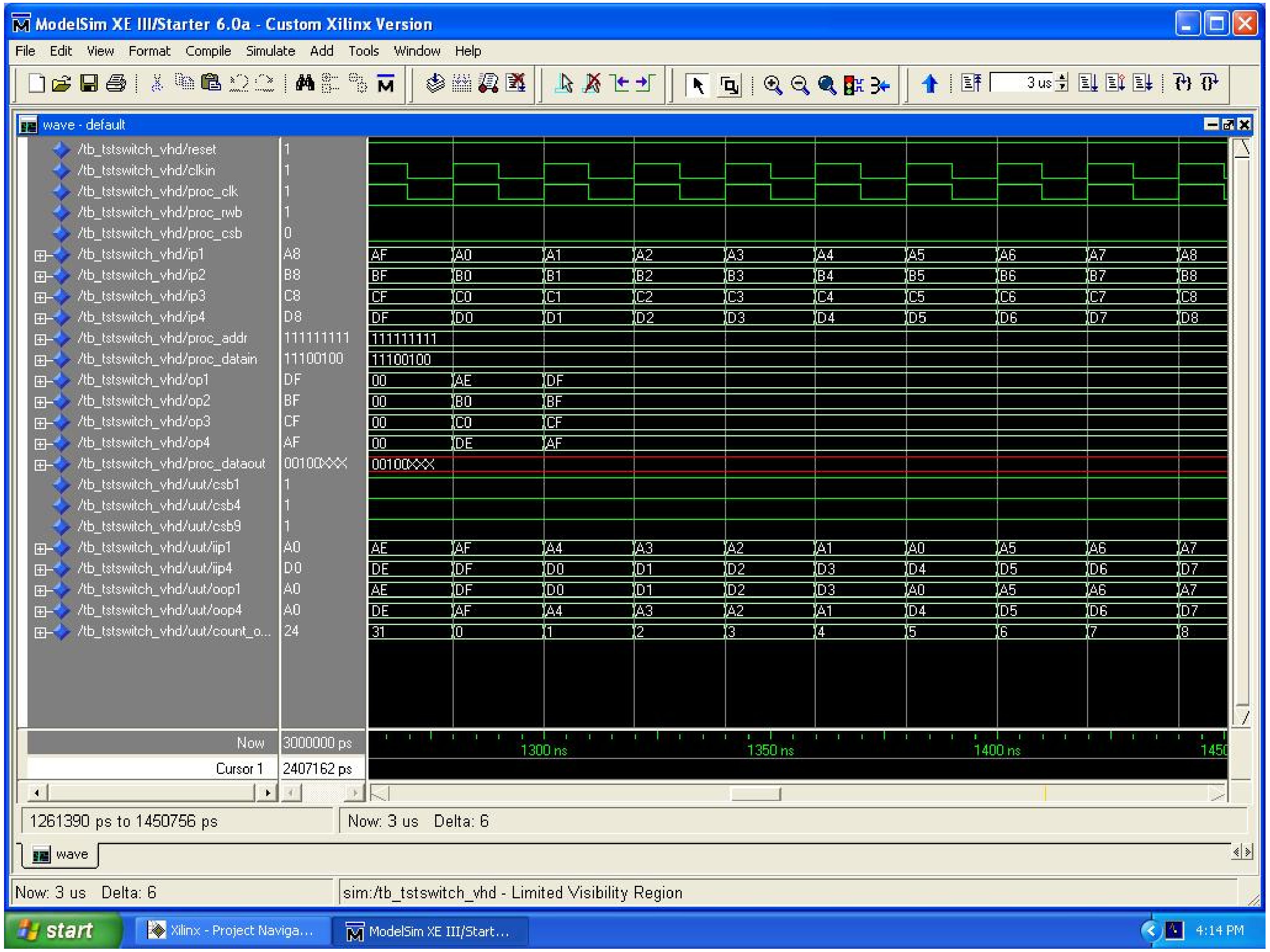
ModelSim XE III/Starter 6.0a - Custom Xilinx Version

wave - default

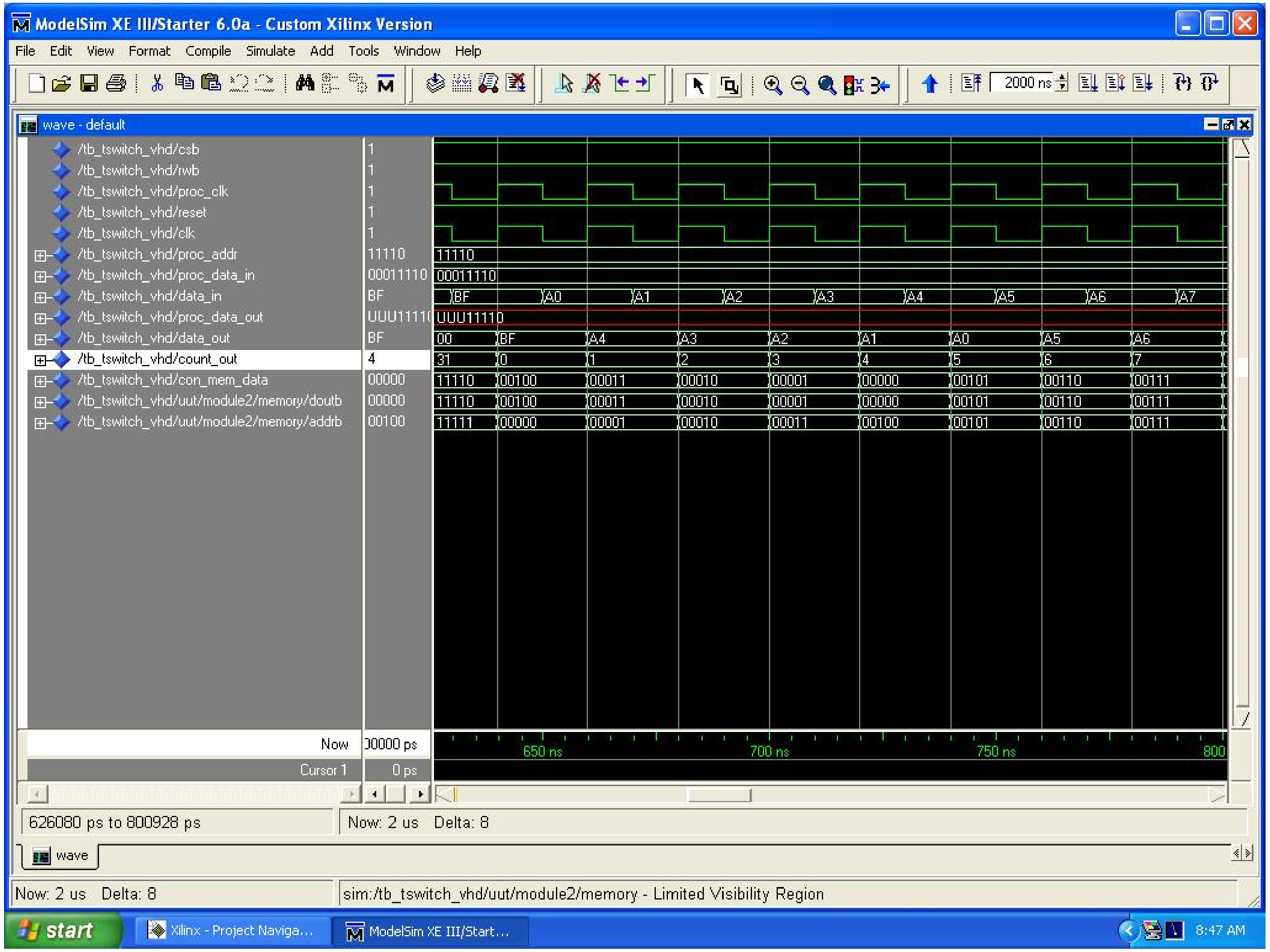
Signal	Value	Waveform
/tb_sswitch_vhd/clk_sw	1	
/tb_sswitch_vhd/rwb_sw	1	
/tb_sswitch_vhd/csb_sw	1	
/tb_sswitch_vhd/clkin	1	
/tb_sswitch_vhd/reset	1	
/tb_sswitch_vhd/addr_sw	11110	11110
/tb_sswitch_vhd/din_sw	11100100	11100100
/tb_sswitch_vhd/ip1	A6	AE AF A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA
/tb_sswitch_vhd/ip2	B6	BE BF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA
/tb_sswitch_vhd/ip3	C6	CE CF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA
/tb_sswitch_vhd/ip4	D6	DE DF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA
/tb_sswitch_vhd/dout_sw	11100100	11100100
/tb_sswitch_vhd/op1	A6	AE AF D0 A1 B2 A3 C4 A5 A6 A7 A8 A9 AA
/tb_sswitch_vhd/op2	A6	BE BF C0 B1 A2 B3 D4 B5 B6 B7 B8 B9 BA
/tb_sswitch_vhd/op3	A6	CE CF B0 C1 D2 C3 A4 C5 C6 C7 C8 C9 CA
/tb_sswitch_vhd/op4	A6	DE DF A0 D1 C2 D3 B4 D5 D6 D7 D8 D9 DA
/tb_sswitch_vhd/count_out	22	30 31 0 1 2 3 4 5 6 7 8 9 10
Now	3000000 ps	00 ns 650 ns 700 ns 750 ns 800 ns 850 n
Cursor 1	0 ps	

597442 ps to 852930 ps   Now: 3 us Delta: 5

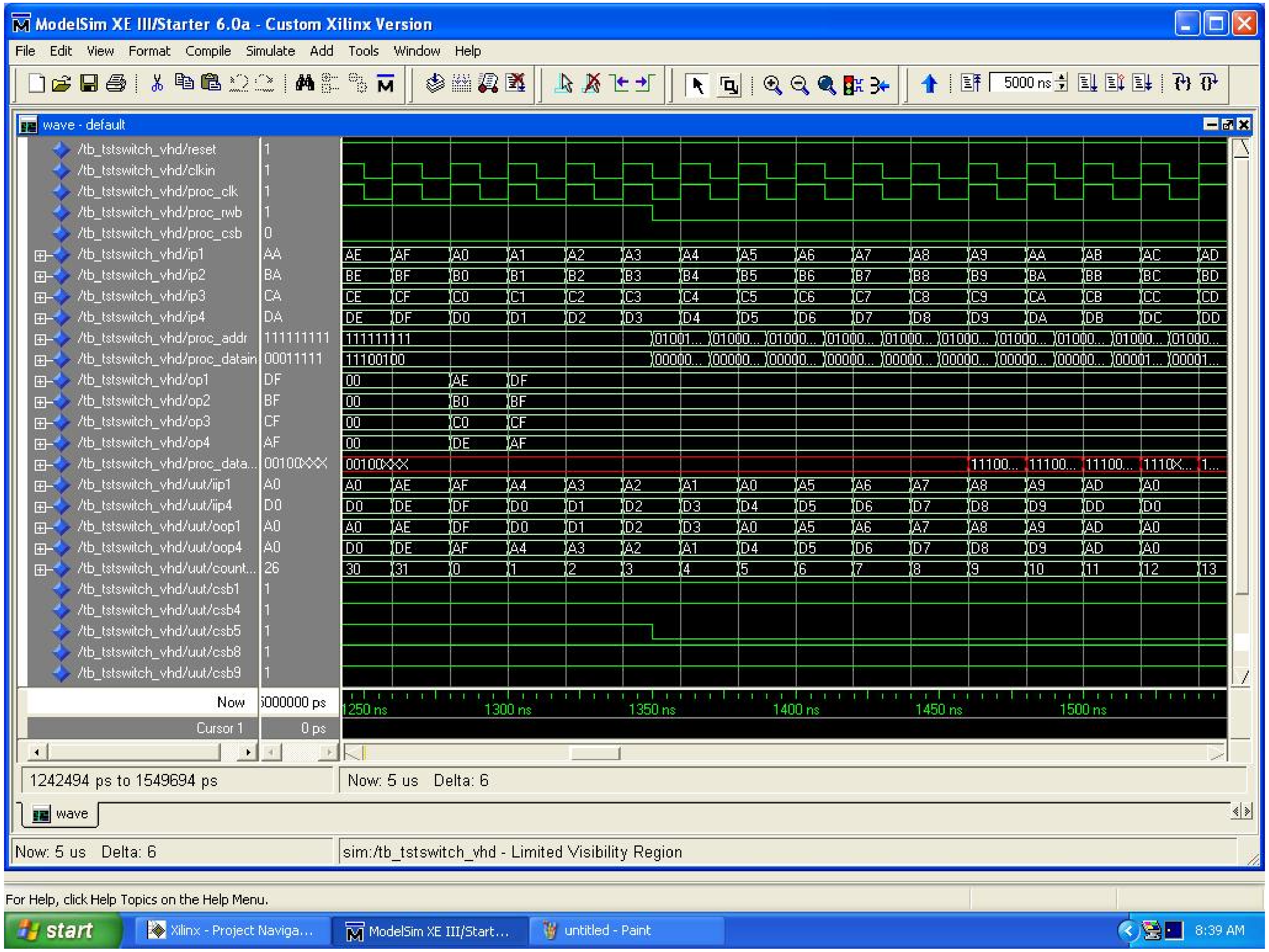
Now: 3 us Delta: 5   sim:/tb_sswitch_vhd - Limited Visibility Region


ModelSim XE III/Starter 6.0a - Custom Xilinx Version
File Edit View Format Compile Simulate Add Tools Window Help
wave - default
wave
start
Xilinx - Project Naviga...
ModelSim XE III/Start...
4:14 PM

```
/tb_tstswitch_vhd/reset              1          
/tb_tstswitch_vhd/clkin              1
/tb_tstswitch_vhd/proc_clk           1
/tb_tstswitch_vhd/proc_rwb           1
/tb_tstswitch_vhd/proc_csb           0
/tb_tstswitch_vhd/ip1                A8         AF A0 A1 A2 A3 A4 A5 A6 A7 A8
/tb_tstswitch_vhd/ip2                B8         BF B0 B1 B2 B3 B4 B5 B6 B7 B8
/tb_tstswitch_vhd/ip3                C8         CF C0 C1 C2 C3 C4 C5 C6 C7 C8
/tb_tstswitch_vhd/ip4                D8         DF D0 D1 D2 D3 D4 D5 D6 D7 D8
/tb_tstswitch_vhd/proc_addr          111111111  111111111
/tb_tstswitch_vhd/proc_datain        11100100   11100100
/tb_tstswitch_vhd/op1                DF         00 AE DF
/tb_tstswitch_vhd/op2                BF         00 B0 BF
/tb_tstswitch_vhd/op3                CF         00 C0 CF
/tb_tstswitch_vhd/op4                AF         00 DE AF
/tb_tstswitch_vhd/proc_dataout       00100XXX   00100XXX
/tb_tstswitch_vhd/uut/csb1           1
/tb_tstswitch_vhd/uut/csb4           1
/tb_tstswitch_vhd/uut/csb9           1
/tb_tstswitch_vhd/uut/iip1           A0         AE AF A4 A3 A2 A1 A0 A5 A6 A7
/tb_tstswitch_vhd/uut/iip4           D0         DE DF D0 D1 D2 D3 D4 D5 D6 D7
/tb_tstswitch_vhd/uut/oop1           A0         AE DF D0 D1 D2 D3 A0 A5 A6 A7
/tb_tstswitch_vhd/uut/oop4           A0         DE AF A4 A3 A2 A1 D4 D5 D6 D7
/tb_tstswitch_vhd/uut/count_o...     24         31 0 1 2 3 4 5 6 7 8

Now       3000000 ps       1300 ns     1350 ns     1400 ns     1450
Cursor 1  2407162 ps

1261390 ps to 1450756 ps          Now: 3 us   Delta: 6

Now: 3 us   Delta: 6              sim:/tb_tstswitch_vhd - Limited Visibility Region
```



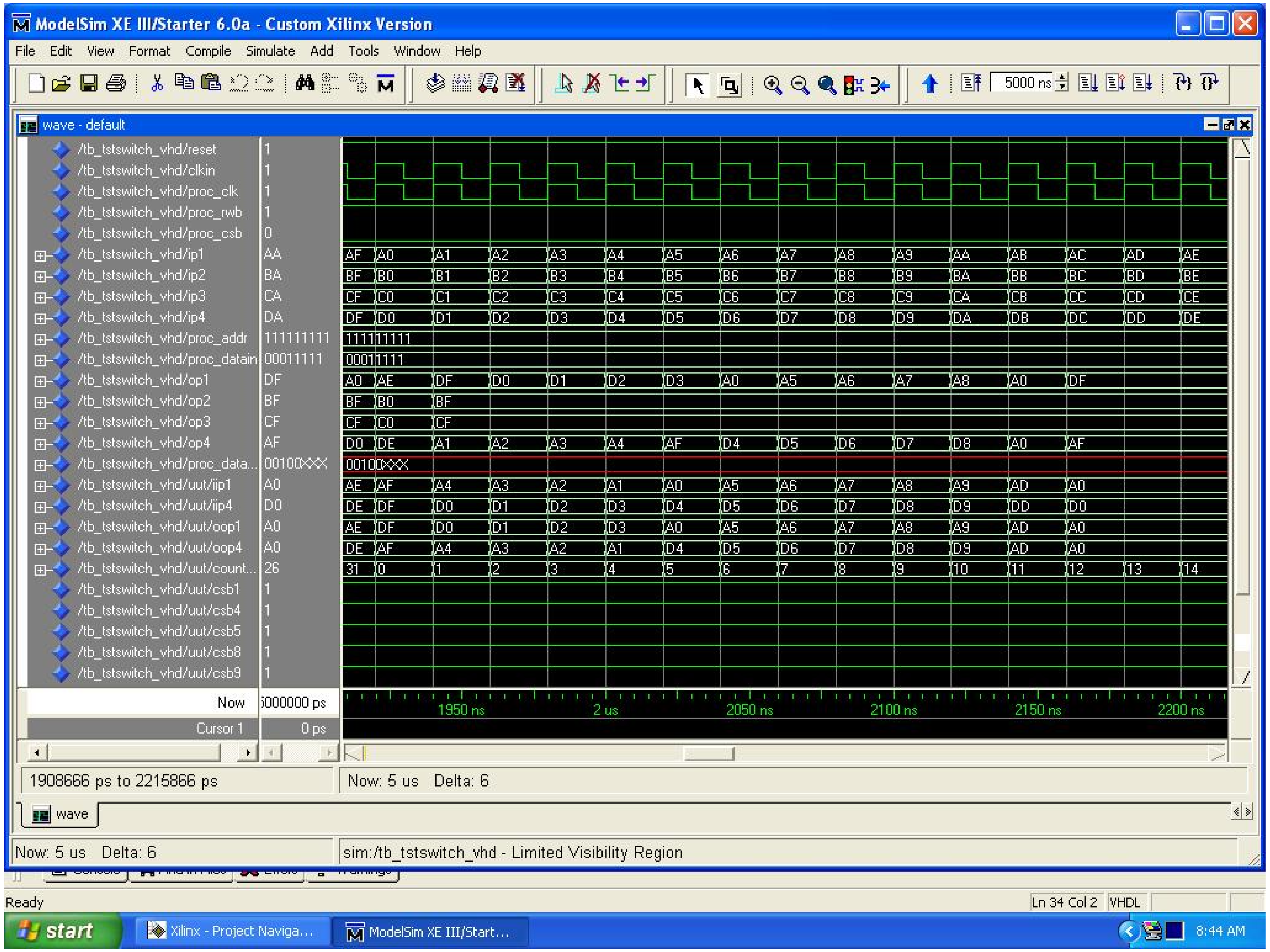
```
ModelSim XE III/Starter 6.0a - Custom Xilinx Version
wave - default

/tb_tswitch_vhd/csb                        1
/tb_tswitch_vhd/rwb                        1
/tb_tswitch_vhd/proc_clk                   1
/tb_tswitch_vhd/reset                      1
/tb_tswitch_vhd/clk                        1
/tb_tswitch_vhd/proc_addr                  11110      11110
/tb_tswitch_vhd/proc_data_in               00011110   00011110
/tb_tswitch_vhd/data_in                    BF         BF    A0    A1    A2    A3    A4    A5    A6    A7
/tb_tswitch_vhd/proc_data_out              UUU11110   UUU11110
/tb_tswitch_vhd/data_out                   BF         00    BF    A4    A3    A2    A1    A0    A5    A6
/tb_tswitch_vhd/count_out                  4          31    0     1     2     3     4     5     6     7
/tb_tswitch_vhd/con_mem_data               00000      11110 00100 00011 00010 00001 00000 00101 00110 00111
/tb_tswitch_vhd/uut/module2/memory/doutb   00000      11110 00100 00011 00010 00001 00000 00101 00110 00111
/tb_tswitch_vhd/uut/module2/memory/addrb   00100      11111 00000 00001 00010 00011 00100 00101 00110 00111

Now        00000 ps        650 ns        700 ns        750 ns        800
Cursor 1   0 ps

626080 ps to 800928 ps     Now: 2 us   Delta: 8
Now: 2 us   Delta: 8       sim:/tb_tswitch_vhd/uut/module2/memory - Limited Visibility Region
```


ModelSim XE III/Starter 6.0a - Custom Xilinx Version

wave - default

```
/tb_tstswitch_vhd/reset          1
/tb_tstswitch_vhd/clkin          1
/tb_tstswitch_vhd/proc_clk       1
/tb_tstswitch_vhd/proc_rwb       1
/tb_tstswitch_vhd/proc_csb       0
/tb_tstswitch_vhd/ip1            AA         AE AF A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD
/tb_tstswitch_vhd/ip2            BA         BE BF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD
/tb_tstswitch_vhd/ip3            CA         CE CF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD
/tb_tstswitch_vhd/ip4            DA         DE DF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD
/tb_tstswitch_vhd/proc_addr      111111111  111111111 01001... 01000... 01000... 01000... 01000... 01000... 01000... 01000... 01000... 01000...
/tb_tstswitch_vhd/proc_datain    00011111   11100100 00000... 00000... 00000... 00000... 00000... 00000... 00000... 00000... 00001... 00001...
/tb_tstswitch_vhd/op1            DF         00 AE DF
/tb_tstswitch_vhd/op2            BF         00 B0 BF
/tb_tstswitch_vhd/op3            CF         00 C0 CF
/tb_tstswitch_vhd/op4            AF         00 DE AF
/tb_tstswitch_vhd/proc_data...   00100XXX   00100XXX 11100... 11100... 11100... 1110X... 1...
/tb_tstswitch_vhd/uut/iip1       A0         A0 AE AF A4 A3 A2 A1 A0 A5 A6 A7 A8 A9 AD A0
/tb_tstswitch_vhd/uut/iip4       D0         D0 DE DF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DD D0
/tb_tstswitch_vhd/uut/oop1       A0         A0 AE DF D0 D1 D2 D3 A0 A5 A6 A7 A8 A9 AD A0
/tb_tstswitch_vhd/uut/oop4       A0         D0 DE AF A4 A3 A2 A1 D4 D5 D6 D7 D8 D9 AD A0
/tb_tstswitch_vhd/uut/count...   26         30 31 0 1 2 3 4 5 6 7 8 9 10 11 12 13
/tb_tstswitch_vhd/uut/csb1       1
/tb_tstswitch_vhd/uut/csb4       1
/tb_tstswitch_vhd/uut/csb5       1
/tb_tstswitch_vhd/uut/csb8       1
/tb_tstswitch_vhd/uut/csb9       1

Now       5000000 ps   1250 ns   1300 ns   1350 ns   1400 ns   1450 ns   1500 ns
Cursor 1  0 ps

1242494 ps to 1549694 ps      Now: 5 us   Delta: 6

Now: 5 us   Delta: 6          sim:/tb_tstswitch_vhd - Limited Visibility Region

For Help, click Help Topics on the Help Menu.
```

File Edit View Format Compile Simulate Add Tools Window Help
5000 ns
wave
start
Xilinx - Project Naviga...
ModelSim XE III/Start...
untitled - Paint
8:39 AM


ModelSim XE III/Starter 6.0a - Custom Xilinx Version
File Edit View Format Compile Simulate Add Tools Window Help
5000 ns
wave - default
1950 ns
2 us
2050 ns
2100 ns
2150 ns
2200 ns
wave
Ready
Ln 34 Col 2
VHDL
start
Xilinx - Project Naviga...
ModelSim XE III/Start...
8:44 AM

```
/tb_tstswitch_vhd/reset            1
/tb_tstswitch_vhd/clkin            1
/tb_tstswitch_vhd/proc_clk         1
/tb_tstswitch_vhd/proc_rwb         1
/tb_tstswitch_vhd/proc_csb         0
/tb_tstswitch_vhd/ip1              AA
/tb_tstswitch_vhd/ip2              BA
/tb_tstswitch_vhd/ip3              CA
/tb_tstswitch_vhd/ip4              DA
/tb_tstswitch_vhd/proc_addr        111111111
/tb_tstswitch_vhd/proc_datain      00011111
/tb_tstswitch_vhd/op1              DF
/tb_tstswitch_vhd/op2              BF
/tb_tstswitch_vhd/op3              CF
/tb_tstswitch_vhd/op4              AF
/tb_tstswitch_vhd/proc_data...     00100XXX
/tb_tstswitch_vhd/uut/iip1         A0
/tb_tstswitch_vhd/uut/iip4         D0
/tb_tstswitch_vhd/uut/oop1         A0
/tb_tstswitch_vhd/uut/oop4         A0
/tb_tstswitch_vhd/uut/count...     26
/tb_tstswitch_vhd/uut/csb1         1
/tb_tstswitch_vhd/uut/csb4         1
/tb_tstswitch_vhd/uut/csb5         1
/tb_tstswitch_vhd/uut/csb8         1
/tb_tstswitch_vhd/uut/csb9         1
Now       5000000 ps
Cursor 1  0 ps
```

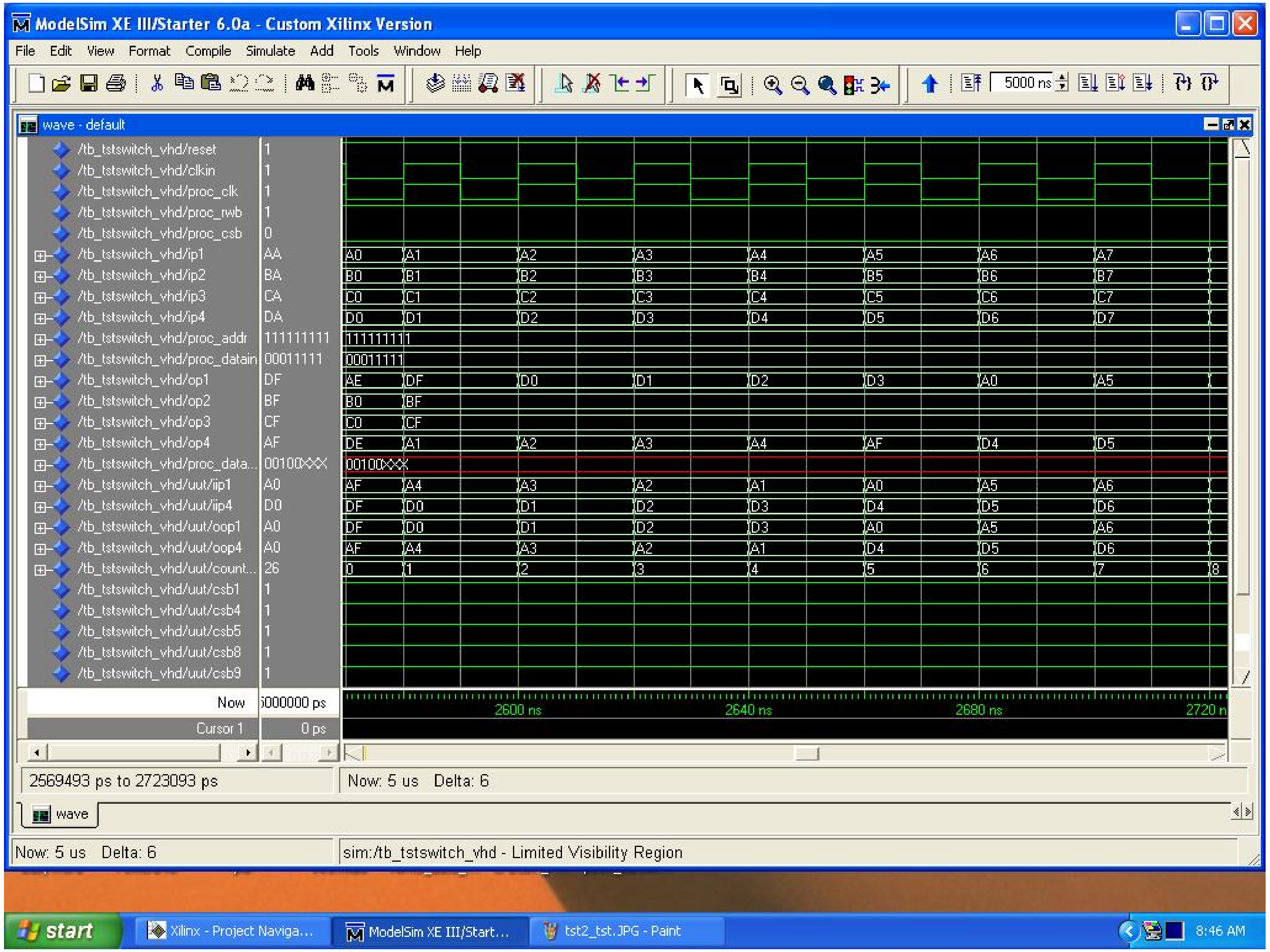
```
ip1        AF A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE
ip2        BF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE
ip3        CF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE
ip4        DF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE
proc_addr  111111111
proc_datain 00011111
op1        A0 AE DF D0 D1 D2 D3 A0 A5 A6 A7 A8 A0 DF
op2        BF B0 BF
op3        CF C0 CF
op4        D0 DE A1 A2 A3 A4 AF D4 D5 D6 D7 D8 A0 AF
proc_data  00100XXX
iip1       AE AF A4 A3 A2 A1 A0 A5 A6 A7 A8 A9 AD A0
iip4       DE DF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DD D0
oop1       AE DF D0 D1 D2 D3 A0 A5 A6 A7 A8 A9 AD A0
oop4       DE AF A4 A3 A2 A1 D4 D5 D6 D7 D8 D9 AD A0
count      31 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

1908666 ps to 2215866 ps

Now: 5 us Delta: 6

Now: 5 us Delta: 6

sim:/tb_tstswitch_vhd - Limited Visibility Region



```
ModelSim XE III/Starter 6.0a - Custom Xilinx Version
wave - default

/tb_tstswitch_vhd/reset          1
/tb_tstswitch_vhd/clkin          1
/tb_tstswitch_vhd/proc_clk       1
/tb_tstswitch_vhd/proc_rwb       1
/tb_tstswitch_vhd/proc_csb       0
/tb_tstswitch_vhd/ip1            AA         A0  A1  A2  A3  A4  A5  A6  A7
/tb_tstswitch_vhd/ip2            BA         B0  B1  B2  B3  B4  B5  B6  B7
/tb_tstswitch_vhd/ip3            CA         C0  C1  C2  C3  C4  C5  C6  C7
/tb_tstswitch_vhd/ip4            DA         D0  D1  D2  D3  D4  D5  D6  D7
/tb_tstswitch_vhd/proc_addr      111111111  111111111
/tb_tstswitch_vhd/proc_datain    00011111   00011111
/tb_tstswitch_vhd/op1            DF         AE  DF  D0  D1  D2  D3  A0  A5
/tb_tstswitch_vhd/op2            BF         B0  BF
/tb_tstswitch_vhd/op3            CF         C0  CF
/tb_tstswitch_vhd/op4            AF         DE  A1  A2  A3  A4  AF  D4  D5
/tb_tstswitch_vhd/proc_data...   00100XXX   00100XXX
/tb_tstswitch_vhd/uut/iip1       A0         AF  A4  A3  A2  A1  A0  A5  A6
/tb_tstswitch_vhd/uut/iip4       D0         DF  D0  D1  D2  D3  D4  D5  D6
/tb_tstswitch_vhd/uut/oop1       A0         DF  D0  D1  D2  D3  A0  A5  A6
/tb_tstswitch_vhd/uut/oop4       A0         AF  A4  A3  A2  A1  D4  D5  D6
/tb_tstswitch_vhd/uut/count...   26         0   1   2   3   4   5   6   7   8
/tb_tstswitch_vhd/uut/csb1       1
/tb_tstswitch_vhd/uut/csb4       1
/tb_tstswitch_vhd/uut/csb5       1
/tb_tstswitch_vhd/uut/csb8       1
/tb_tstswitch_vhd/uut/csb9       1

Now       5000000 ps        2600 ns     2640 ns     2680 ns     2720 n
Cursor 1  0 ps

2569493 ps to 2723093 ps    Now: 5 us   Delta: 6

Now: 5 us   Delta: 6    sim:/tb_tstswitch_vhd - Limited Visibility Region
```

6. PRACTICAL UTILITY AND FUTURE ENHANCEMENTS

6.1 IMPLEMENTED ADPLL

Implemented ADPLL is a very optimize circuit, hardware wise. This type of circuit can practically be used anywhere, where there is need of locking clock with standard reference clock. This type of ADPLL is very useful when the PLL is to be fabricated inside an Integrated Circuit or to be programmed inside an FPGA or PLD.

Presently the ADPLL is designed in keeping standard E1 (2.048 Mbps) bit rate for PCM frames. For this frequency, a standard high frequency oscillator of 65.536 MHz is used (2.048 x 2^6 = 65.536).Thus registers in loop filter are chosen to be of 6 bits each. Maximum output jitter in this type of ADPLL is one clock period of standard high frequency clock.

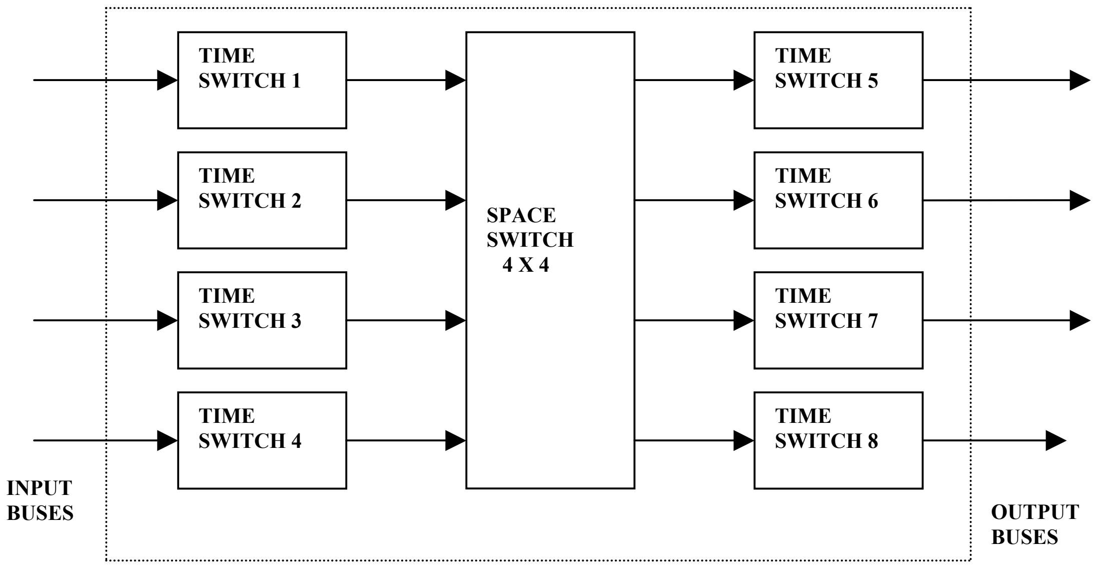
Future enhancements can be done by upgrading this design for other standard bit rates also, as is already done for 8.192 Mbps in the present design. Moreover maximum output jitter can also be reduced by using higher bit rate standard clock.

For this only register bits of the registers of loop filter are to be increased and rest of the design will remain same. Number of register bits can be calculated from the formula:

(output clock rate) x $2^{(\text{no. of register bits})}$ = Standard high frequency clock rate.

6.2 IMPLEMENTED TST SWITCH

By the elements "time switch" and "space switch" a complete TST switch is implemented.



IMPLEMENTED TST SWITCH

Presently the time and space switches are designed according to PCM frames i.e., 32 bytes per frame. This design can be used in any system where there is need to interchange time slots in PCM frame at 2.048 Mbps rate. This design can also be used while mapping E1 (2.048 Mbps) tributaries on higher bit rate synchronous frames. This design can also be used in ISDN systems, after the line interface unit, for switching or interchanging time slots.

This system can be enhanced for more number of buses (presently it is for four input and four output buses only).

This system can also be enhanced for larger frame sizes and other bit rate signals.

These enhancements can easily be implemented in present framework of VHDL code. For more number of buses, more time switch units are needed and capacity of space switch is to be increased. For larger frames, memory requirements at time and space switch units will be more.

BIBLIOGRAPHY :

- Phase Locked Loops, Theory, Design & Applications – by Roland E Best, *McGraw Hill, Inc.*
- IEEE journals, solid state circuits, vol 34. No 8, aug 99
- IEEE transactions on Circuit and systems-II, Analog & Digital signal processing, vol 46, No 7, july99
- *IEEE proceedings*; vol 63, no 2, feb 1975
- W.C.Lindsey, F.Ghazvinian, W.C.Hagmann, K. Dessouky. Network Synchronization. *Proceedings of the IEEE 1985*
- P. Kartaschoff. Synchronization in digital communication networks. *Proceedings of the IEEE 1991*
- S. Bregni. A historical perspective on network synchronization. *IEEE Communication Magazine 1998*
- Telecommunication Switching Systems and Networks – by Thiagarajan Viswanathan, *Prentice Hall fo India Pvt. Ltd.*
- Amose E. Joel, Digital Switching – How it has developed. IEEE transaction on *communications, Vol. Com-27, No.7, July 1979*
- Synchronization of Digital Telecommunication Networks – by Stefano Bregni, *John Wiley & Sons, Ltd.*
- http://www.agilent.com
- http://www.xilinx.com

TOOLS USED

- Altera MaxPlus-II ver 10.2
- Modelsim XE-II 5.7g
- Xilinx ISE 5.7i