Abstract

BHARATH, BHASKAR. Symbol Recovery Circuit design for deep-space MARS receiver using SOI technology(Under the direction of Dr. Paul Franzon)

The motivation for this thesis is to present a design of a symbol recovery circuit for a receiver on a planetary lander vehicle, which will communicate with a low orbit satellite. With the development of advanced propulsion mechanisms and autonomous machines, there has been a great revival of interest in exploration of nearby planets. The Mars rover project aims to land an autonomous vehicle on the surface of Mars, which would be controlled via a datalink with an orbiting satellite. The design of a communication system for this presents a number of issues, including Doppler resistance, Radiation tolerance and minimal power consumption.

The design of the receiver uses a new modulation technique known as Double Differential Phase Shift Keying, which provides the inherent robustness to Doppler while consuming low power. A symbol recovery circuit is an essential part of the receiver and extracts clock information from received data which is then used to demodulate data. The symbol recovery circuit for the rover needs to handle multiple bit rates while consuming minimal power. The design of the receiver on a system level and the symbol recovery circuit is described in the thesis. The system was modeled using MATLAB simulink, designed in Verilog/VHDL, synthesized using Synopsys design compiler and converted to SOI using Cadence.

Symbol Recovery Circuit design for deep-space MARS receiver using SOI technology

by

Bhaskar Bharath

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Electrical Engineering

Raleigh

2003

Approved By:

Dr. Keith Townsend

Dr. Rhett Davis

Dr. Paul Franzon
Chair of Advisory Committee

To my parents

Biography

Bhaskar Bharath was born in Salem, India in October 1979. He graduated from Indian Institute of Technology, Madras, with a Bachelors degree in Electrical Engineering, in July 2001. In August 2001, He joined the Masters program in the department of Electrical and Computer Engineering at North Carolina State University, Raleigh, NC. While working towards the Masters degree, he worked on his thesis under the guidance of Dr.Paul Franzon.

Acknowledgements

I wish to express my sincere gratitude to my thesis advisor, Dr Paul Franzon, for providing me a great opportunity to work and learn under him. It was a great experience working under his guidance. I wish to thank Dr Rhett Davis who has provided me guidance and has helped me in many issues in my thesis. I would like to thank Dr Keith Townsend for providing me support and guidance throughout my study here. I would like to thank Mehmet Yuce and John Damiano for their help and guidance, which went a long way in helping me do my thesis work.

I am greatly indebted to my parents for everything they have done for me. Without their love and support, I would never have been able to succeed in my endeavors. I would like to thank Savitha for helping me and providing me moral support through my thesis. I would like to thank my friends Sudarshan, Anand, Srikanth and Sandeep for helping my through tough times and making my stay here worthwhile.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Space communication via satellites has led to a rapid increase in connectivity throughout the world. Space based systems find diverse applications in a several areas including meteorology, geology, wireless applications and defense. The amount of data processing in these systems has been rising steadily. Wireless space receivers rely on solar energy for their power needs and hence need to be highly power efficient. Apart from that, they need to be small and light weight, due to the limited space available in a satellite. The environment in space is highly prone to cosmic radiation and temperature variation from the sun and hence satellite-based systems need to be robust to it.

Satellites are of two types, geo-stationary satellites which are stationary with respect to the ground (they move at the same speed as the planet rotates) and Low-orbit satellites (non-geostationary). The latter move at lower orbits and revolve at velocities different from the planet's rotation speed. Design of receivers for such satellites is further complicated due to the presence of Doppler shifts. An accelerating satellite leads to a changing Doppler which affects the received and transmitted data, making design of receiver complicated.

A design of a receiver targeted for the next generation planetary lander, which communicates with a low-orbit satellite under such circumstances, is proposed here. The system is designed to support a large number of bit-rates ranging from 100 Kbps to 100 bps while being robust in a channel containing Doppler shifts and additive white Gaussian noise. The power consumption of the receiver (in the lander) is more critical than that of the transmitter (in the orbiter). This asymmetry makes it possible to trade off transmit power against the power consumption in the receiver. The project's overall objective is to design a single chip receiver suitable for Mars orbiter to lander communications with the lowest possible power consumption. Honeywell's Rad-hard SOI technology is used to make it robust to radiation and temperature variations.

The design of the symbol recovery circuit for the receiver is also presented here. The symbol recovery circuit needs to handle a variety of bit-rates while maintaining synchronization with the transmitter at all times. A system level design for the receiver and a circuit level design of the multi-bit rate clock recovery system are presented here. The clock recovery system consists of an all-digital phase locked loop (ADPLL) for multi-bit rate applications. The receiver works at a carrier frequency of 431.7 MHz using a Double Differential PSK scheme to modulate the data. The supported bit-rates are 100 Kbps, 10 Kbps, 1 Kbps and 100 bps. The emphasis of the design is on power reduction and Doppler robustness.

1.2 Organization of Thesis

Chapter 2 describes various techniques used in the design of space receivers, with emphasis on Doppler-resistance and energy efficiency. Chapter 3 covers various methods of clock/symbol recovery using phase locked loops. Chapter 4 describes the proposed receiver on a system level and analyzes the impact of architectural changes. Chapter 5 describes the hardware design of the clock recovery circuit. It describes the design methodology, working and testing procedure used. Chapter 6 concludes with a discussion of the results and future work.

Chapter 2

Design of Space receivers

2.1 Introduction

Design of receivers is determined by a combination of factors, the most important being the nature of the channel and cost. In space communication, data transfer between satellites and between satellites and ground-based station is crucial. Unlike ground based systems, which are slow moving, satellites revolve around the earth at relatively high velocities, which results in a constantly varying channel and leads to Doppler variations in the carrier.

The Mars Lander program aims at landing a mobile robot on the ground which is in communication with a satellite in low orbit around Mars. The LEO or the Low earth orbit is a very well studied channel and its treatment is complex due to interference effects in the atmosphere, motion of the satellite around the planet and due to fading effects in the earth atmosphere [1, 2, 3, 4, 5, 6, 7]. Martian atmosphere is much rarer than Earth's and does not have interference from other satellites. This makes the treatment of a Low Mars orbit channel much simpler [19].

In this chapter we describe various techniques used to design space based communication receivers. Various modulation techniques used to design space communication systems are described. Later, an example FSK space receiver is described.

2.2 Modulation Techniques for space applications

In a variety of applications of digital communications, the channel introduces a random frequency shift in the carrier which is difficult to estimate on the basis of its previous history. An example of such a channel is the random Doppler frequency shift communication system often found in space based applications. The Doppler shift channel implies that the modulating carrier frequency as seen by the ground-based receiver keeps varying. Due to this, it is not possible to use a coherent detection scheme, where the carrier frequency needs to be well-defined. Hence, one must resort to a form of incoherent detection, which includes the differential and double-differential detection.

The most common modulation scheme used in space receivers is Frequency Shift Keying or FSK. FSK demodulators are very simple in design, and hence minimize power and area, which are critical for space applications. However, in the presence of Doppler shifts, FSK performance deteriorates and additional circuitry is needed to compensate for frequency variations. There are a variety of modulation techniques which are inherently Doppler resistant and provide promising alternatives to FSK. One of these is Double Differential Phase Shift Keying (DDPSK) which uses the second order difference to compensate for frequency variations in the carrier.

In this section we discuss in detail two techniques, Non-coherent FSK and Double differential PSK. The focus is on application to space communication systems and on a low-power, low-area architecture which is demanded by such systems.

2.2.1 Frequency Shift Keying and FDMA

Frequency Shift Keying [8, 9] is the most commonly used form of digital modulation and leads to a simple receiver design. A phase locked loop can be used to demodulate FSK signals, as information is transmitted as frequency variation, which is very well tracked by a PLL. In an FSK system symbols are distinguished from each other by associating a different frequency with each symbol. In our discussion here, we focus primarily on the binary frequency shift keying or BFSK. In this the symbols 1 and 0 are associated with two different frequencies. An example of a BFSK signal is shown in figure 2.1. A typical pair of such signals would be,

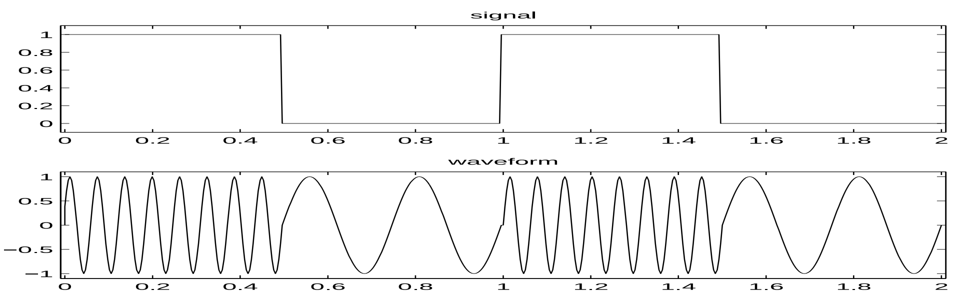

Figure 2.1: Frequency Shift Keying (FSK)

$$s_i(t) = \begin{cases} \sqrt{\frac{2E}{T}} cos(2\pi f_i t) & 0 \leq t \leq T_b \\ 0 & \text{elsewhere.} \end{cases}$$

Where i=1, 2 and E is the energy per bit and T is the bit duration. The transmitted frequency is given by,

$$f_i = \frac{n+i}{T} \text{ for some fixed integer n, i = 1,2}$$

The most commonly used binary FSK system is the Sunde's FSK. It is a continuous-phase signal in the sense that the phase continuity is always maintained at the transitions.

Encoding

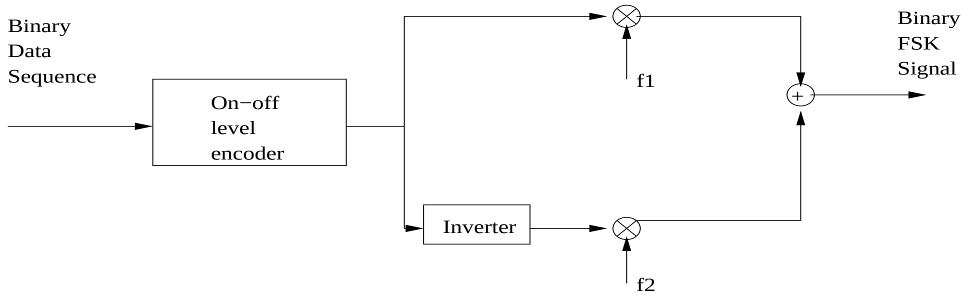

Figure 2.2: FSK encoder: functional block diagram

The design of an encoder for FSK is very simple due to the inherent nature of the modulation scheme. Figure 2.2 shows the basic block diagram of an FSK encoder known as the on-off encoder. The incoming data sequence is applied to an on-off level encoder which gives a value of $\sqrt{E}$ to the symbol 1 and 0 to the symbol 0. This is then fed to an oscillator at frequency f_1 and another oscillator at frequency f_2 through an inverter. This ensures that only one oscillator is on at any given instant. The output is added to get the resulting FSK signal.

Decoding

The demodulation techniques for FSK signals fall into two main categories: FM detector demodulators and filter-type demodulators. These are discussed below

1. *FM detector demodulators*

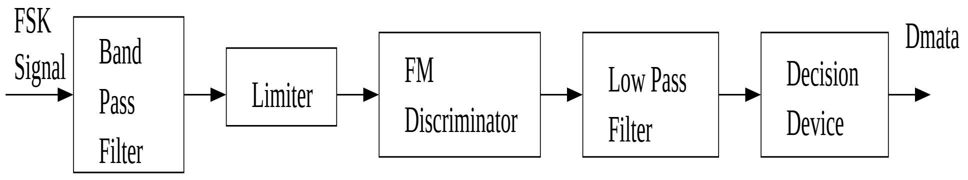

Figure 2.3: FSK FM detector: functional block diagram

FM detector demodulators treat the FSK signal as an FM signal with binary data. These demodulators are the traditional method for recovering FSK data from the modulated signal. Figure 2.3 shows the block diagram of a FM detector demodulator. The input signal is band-pass filtered and limited and then passed through an FM discriminator. The FM discriminator converts the high frequency modulated signal into a raw, detected low pass signal. This signal is then filtered and passed through a decision device to get the output data. Phase-locked loop demodulators are a more recent technique based on the FM detectors. Phase locked loops perform superior to the FM detectors and are more commonly used.

2. Filter-type demodulator

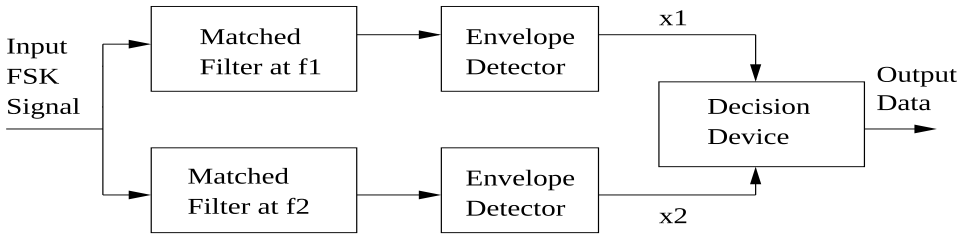

Figure 2.4: FSK filter detector: functional block diagram

The FM detector demodulator though simple in design is not optimal in spectral efficiency. The filter-type demodulators try to overcome this by using a matched-filter arrangement which maximizes spectral efficiency and minimizes bit errors. In this technique the data is passed through a couple of matched filters designed for the frequencies of the FSK signal. A decision is made based on the result of the filters. Figure 2.4 shows an arrangement for FSK demodulation using this technique. The filters are designed with the channel and signal and mind.

Error Performance

We discuss briefly the error performance of the FSK matched filter non-coherent detector as shown in figure 2.4. x_1 and x_2 represent the outputs of the two matched filters after envelope detection. When a '1'(frequency f_2) is sent, the detector makes an error is $x_1 > x_2$. Therefore, the conditional bit error probability is given by,

$$\begin{aligned} p_e(1) &= \text{Prob}(x_1 > x_2) \\ &= \int_{x_2=0}^{\infty} g_{x_1}(x_1) \int_{x_1=x_2}^{\infty} f_{x_2}(x_2) dx_1 dx_2 \end{aligned}$$

Where, f(x) is a Raleigh distribution on x and g(x) is a Rician distribution on x. Evaluating the integral this leads to $p_e(1) = \frac{1}{2}e^{-\frac{E_b}{2N_0}}$. Using a similar procedure for the probability of error when 0 is transmitted it can be shown the $p_e(0) = p_e(1)$. Since the symbols 1 and 0 are equally likely, the average probability of error is given by:

$$\begin{aligned} p_e &= \frac{1}{2}p_e(1) + \frac{1}{2}p_e(0) \\ &= \frac{1}{2}e^{-\frac{E_b}{2N_0}} \end{aligned}$$

Drawbacks

FSK is a simple modulation scheme and leads to an efficient demodulator design in terms of power and area. However, the performance the FSK signal is inferior to other modulation techniques like the PSK. This is because the FSK constellation is spaced closer when compared to BPSK or QPSK. The bandwidth required for an FSK is also larger due to side-lobes and interference. The performance of FSK degrades when used in an environment where the carrier frequency changes. Under such conditions, the receiver needs additional circuitry to tackle the variations in the carrier [10, 19].

2.2.2 Double-differential phase shift keying

Phase shift keying is a modulation scheme in which the signal is modulated in the phase of the carrier. A symbol 1 implies a phase of 0^o while a symbol 0 corresponds to a phase of 180^o. To decode a PSK signal we need to know the carrier frequency and it limits its practical applicability. To alleviate the problem a differential detection is used.

However, classic differential detection (encoding the data phase information as a first-order difference phase process) is sensitive to frequency variation of carrier. In particular if $\Delta\omega = 2\pi\Delta f$ denotes the radian shift in carrier frequency introduced in the channel, and $\frac{1}{T}$ denotes the data rate, then the detected phase at the output of the differential detector is shifted by $\Delta\omega T$, which if large can have an appreciable effect on performance.

One solution to the above problem is to encode the data as a second-order phase difference and use a two-stage differential detection. This is the principle behind the double differential scheme. Depending on the specific implementation, the output will either be independent of the input frequency or a function of frequency within tolerable limits. A brief description of Double-differential encoding follows [11, 12, 13, 14, 15], Consider a PSK

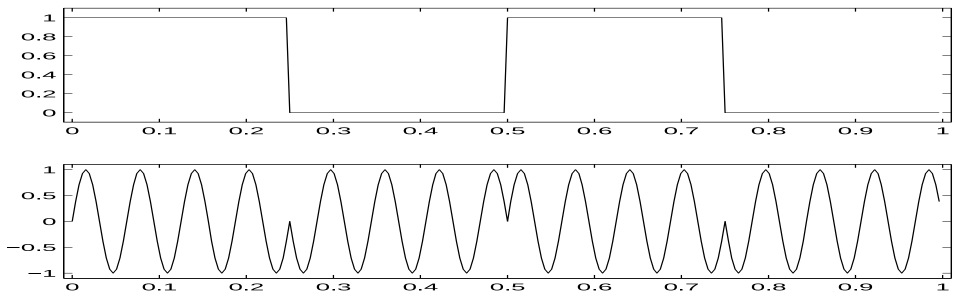

Figure 2.5: Phase Shift Keying (PSK)

signal (Figure 2.5) of the form,

$$s_i(t) = \sqrt{\frac{2E}{T}} \cos(\omega_0 t + \theta_i),\ \text{(i-1)T} < \text{t} \leq \text{iT}$$

where, $s_i(t)$ denotes the signal transmitted over the i^{th} symbol interval, and θ_i is the transmitted signal phase during the i^{th} symbol interval. For an M-ary communication system with the data encoded in the phase angles, the digital data d_i are represented by phase random variables

$$\theta_i \epsilon \Theta(t) = \{\theta : \theta = \frac{2\pi j}{M}, j = 0, 1, \ldots, M-1\}, i \geq 3$$

where, the superscript i indicates that this datum will be encoded in the i^{th} symbol interval.We use the first two symbol intervals to establish a phase reference by letting θ_1=θ_2=θ_{ref}. The phase data is then encoded using a second-order difference equation.

$$\bigtriangledown\theta = \Delta^2\theta_i \bmod 2\pi = \theta_i - 2\theta_{i-1} + \theta_{i-2}, i \geq 3$$

where Δ denotes the difference in phase $\Delta_i \equiv \theta_i - \theta_{i-1}$ and where $\Delta^2\theta \equiv \Delta(\Delta\theta_i)$. The i^{th} transmitted signal phase thus becomes

$$\theta_i = \bigtriangledown\theta - 2\theta_{i-1} + \theta_{i-2} \bmod 2\pi, i \geq 3$$

For the case of M=2(DDPSK), the differential encoding simplifies to

$$\theta_i = \bigtriangledown\theta - \theta_{i-2} \bmod 2\pi, i \geq 3$$

since $2\theta_{i-1} = 0$ module 2π, independent of θ_{i-1} which is either 0 or π. This is the basic principle of a DDPSK encoder.

Encoding

A functional block diagram for the required transmitter configuration is as shown in figure 2.6 and figure 2.7 We illustrate second-order differential encoding with an example

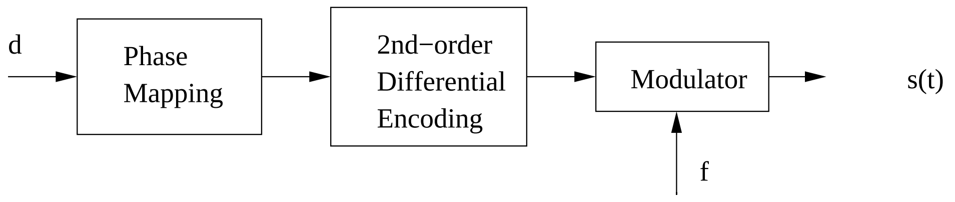

Figure 2.6: DDPSK Encoder: functional block diagram

using a reference angle $\theta_{ref} = \pi$ and $\theta_{ref} = 0$ for the binary scheme. The data stream and the results of the second order differential phase encoding of the data stream d :

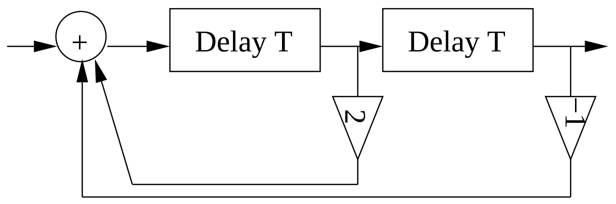

Figure 2.7: Double Differential Encoder: functional block diagram

$1, 0, 1, 1, 1, 0, 0, 1$ are shown in tables 2.1 and table 2.2.It can be seen that above operation is equivalent to xoring the data bits i.e, $d_i = d^i$ xor d_{i-2}.

i	1	2	3	4	5	6	7	8	9	10
d^i			1	0	1	1	1	0	0	1
$\bigtriangledown\theta$			π	0	π	π	π	0	0	π
θ_i	π	π	0	π	π	0	0	0	0	π

Table 2.1: Encoding with $d = 1$ to phase $\bigtriangledown\theta = \pi$ and the $d = 0$ to phase $\bigtriangledown\theta = 0$ with initial phase π

i	1	2	3	4	5	6	7	8	9	10
d^i			1	0	1	1	1	0	0	1
$\bigtriangledown\theta$			π	0	π	π	π	0	0	π
θ_i	0	0	π	0	0	π	π	π	π	0

Table 2.2: Encoding with $d = 1$ to phase $\bigtriangledown\theta = \pi$ and the $d = 0$ to phase $\bigtriangledown\theta = 0$ with initial phase 0

Decoding

The detection of DDPSK signals is similar to the PSK signal except that there are two levels of difference operations to be performed. There is a large collection of literature on the demodulation a DDPSK signal [13, 16, 17]

1. Coherent Detection

   Figure 2.8 shows the functional diagram of a coherent DDPSK detector. This detector needs to know accurately the carrier frequency of the signal and hence is very limited

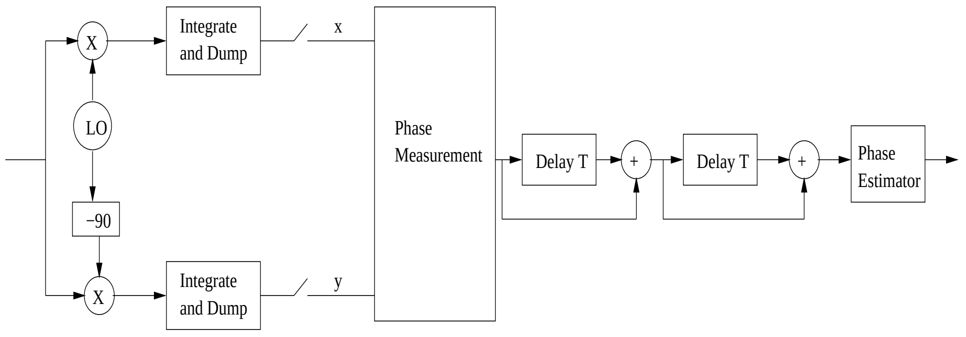

Figure 2.8: DDPSK coherent detector: functional block diagram

in practical applications. The demodulator consists of integrate and dump filters whose output is passed to a phase measurement system. The phase of the signal is given by $\theta = \tan^{-1}(\frac{y}{x})$. The phase is then passed through two differences to get the final phase. A decision device is used to distinguish the symbols.

2. Matched Filter - Autocorrelation detector

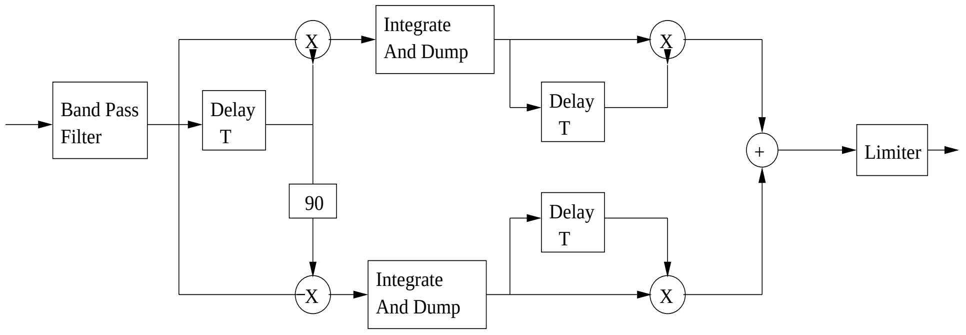

Figure 2.9: DDPSK autocorrelation detector: functional block diagram

The figure 2.9 shows an autocorrelation detector used for demodulating DDPSK signals. The first stage of the demodulator converts the frequency error into a phase error by using a matched filter autocorellator. The second stage then converts this phase error to the required data. This modulator has the advantage of not needing the input carrier signal for demodulation and hence is used commonly for detection of DDPSK signals.

3. Frequency insensitive multiple symbol autocorrelation demodulator

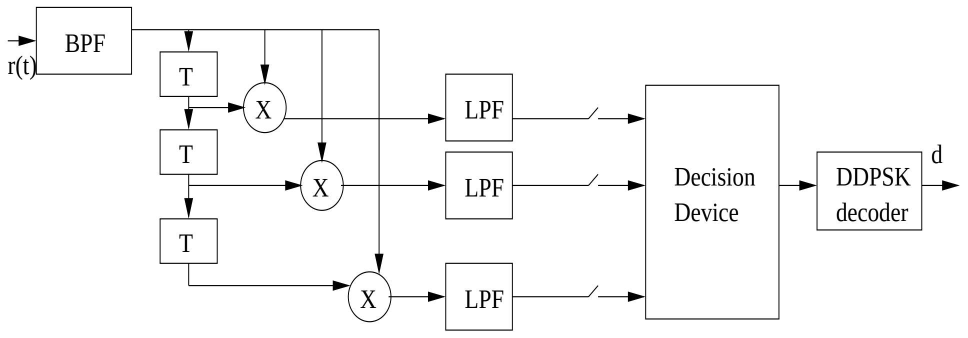

Figure 2.10: Frequency insensitive multiple symbol autocorrelation demodulator

This demodulation scheme uses maximum likelihood decoding to demodulate the DDPSK signal. Figure 2.10 shows the block diagram of such a decoder. The detector decodes a bit stream of L symbols by calculating the autocorrelation and cross-correlation components and choosing the output bits depending on the maximum values of correlation product given by $A_{r,s}(l) = \Sigma_{k=l}^{L-1} \rho_{l,k} s_k^* s_{k-1}$ where $\rho_{l,k} = r_k r_{k-l}^*$ represents the autocorrelation of the received signal with symbol spaced lags l. A detailed treatment of this receiver is presented in [17]. An advantage of this method is multiple symbol detection which leads to better performance over the other receivers at the cost of increased complexity.

Performance

Calculating the probability of error for a DDPSK demodulator is non-trivial problem, due to its complex nature [13, 15]. It can be shown that the performance of the DDPSK detector under ideal conditions is similar to that of the corresponding non-coherent FSK receiver and is 3dB below the performance of a DPSK signal. However, under the presence of frequency errors DDPSK out-performs both DPSK and FSK.

Drawbacks

In DDPSK data is coded as a second order phase difference and decoded by a double differential detection at the receiver. Because of this, the performance of DDPSK is considerably degraded (an order of 3-4 dB) relative to the differential (first-order) and PSK detection. However, in a channel susceptible to carrier frequency variation, differential modulation performance degrades considerably due to frequency sensitivity. DDPSK is inherently frequency insensitive due to which its performance degradation is minimal.

The DDPSK receiver offers a simple architecture and hence is preferable over other schemes in channels when frequency insensitivity (e.g., Doppler shift channel in space) is important. In particular, it lends to a simple clock recovery scheme as the PSK signal is subject to changes in Phase which can be detected easily using a PLL.

2.3 Receiver architecture

A digital communication receiver [18] converts the incoming analog modulated signal into data. The design of a receiver depends on several factors, like cost, power, channel characteristics and area on a die. A receiver consists of the following components as shown in figure 2.11:

1. Antenna

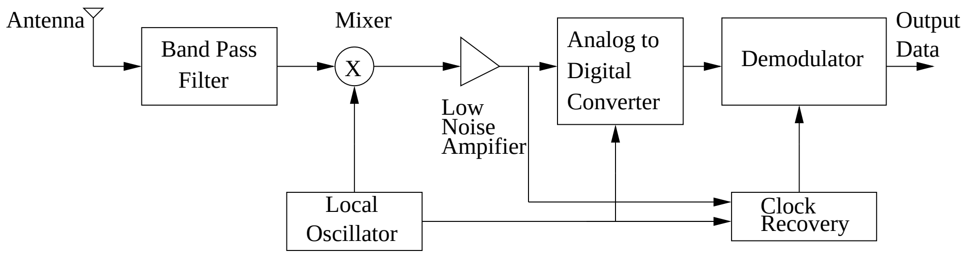

Figure 2.11: Digital Communication Receiver

The modulated signal is propagated in space in the form of electro-magnetic radiation. This needs to be converted back to an electrical signal, so that further processing can be done on it. The antenna converts the weak electro-magnetic signal into an electrical signal.

2. Band Pass Filter

   A transmitted signal consists of a modulating carrier with the modulated data in it. The data is usually contained in a bandwidth around the carrier. The rest of the spectrum contains noise and interference from other sources and needs to be removed. The band pass filter is usually an analog circuit which reduces the noise and interference from other sources by band-limiting the received signal.

3. Low Noise Amplifier (LNA)

   The LNA or Low noise amplifier amplifies the weak signal from the antenna into a large signal to enable better detection and processing. In doing so, the amount of noise added by the amplifier needs to be minimized. The LNA is an analog amplifier and often consumes a large amount of power.

4. Mixer

The mixer is a circuit which converts the high frequency modulated carrier to a lower frequency or to a baseband signal. Most modulating carrier frequencies are too high and designing circuits at those frequencies is not practical. The mixer converts the modulated signal at the carrier frequency, to a lower known frequency, simplifying the design of the receiver.

5. Analog to Digital converter (A/D)

   An Analog to Digital converter is used in digital receivers to convert the analog output of the LNA to a digital signal which can be processed using a computer or a DSP. Digital signals provide the advantage of being easier to manipulate and process. The analog to digital converter can convert the analog signal to a given number of bits. The more number of bits, less the quantization noise added by the A/D to the incoming signal and more the power consumption.

6. Demodulator

   The demodulator converts the output of A/D converter into a decoded data stream. The demodulator can perform a variety of additional functions like frequency correction and estimation.

7. Symbol Recovery Circuit

   Digital data is always processed with an implicit clock. This clock rate needs to be accurately determined at the receiver to decode the information bits correctly. Most modulation schemes have encoded clock information in them which is extracted by the Symbol recovery circuit.

8. Oscillator

   Receivers need to have some internal clock to be able to produce other clocks and demodulate the carrier. The Oscillator used is mostly a crystal oscillator, though a ring oscillator is used in some cases.

2.4 Requirements of the Mars receiver

The Mars receiver is specifically targeted for communications between an orbiting ship and a planetary landing vehicle. This scenario has three important ramifications that motivate design of the receiver.

1. The power consumption of the receiver on the lander is much more critical than the transmission power from the orbiter.

2. No adjacent channel interferes exist, and all noise is additive white Gaussian.

3. The receiver must be able to tolerate large and rapidly changing frequency offsets due to Doppler.

These requirements are summarized in Table 2.3.

Data Rate	Fb	100-1000-10000 Bps
Carrier Frequency	Fc	437.1 MHz
Doppler offset	F_dop	10kHz
Rate of change of Doppler		10 Hz/s

Table 2.3: Requirements of the Space receiver

2.5 Space Receiver Example: UCLA FSK receiver

A space receiver architecture for the above specifications has been designed by UCLA [19] and is shown in figure 2.12. In their implementation of the receiver they have used frequency-shift keying (FSK) as the modulation scheme. The receiver utilizes sub-sampling and 1-bit data processing together with a discrete Fourier transform-based detection scheme to reduce power consumption dramatically compared to other FSK architectures. The salient features of their architecture are:

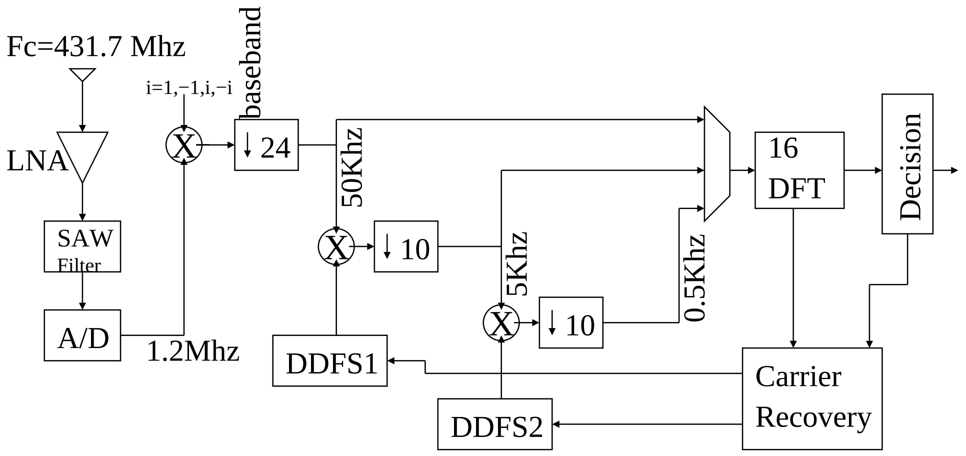

Figure 2.12: FSK receiver

- Sub-sampling

The complete form of the Nyquist criterion, applicable to bandpass signals [19], states that the signal must be sampled at a frequency at least twice the bandwidth (BW) of the signal (just 20 kHz for the UCLA design) even if the signal power is centered around a much higher frequency. This approach results in significant power savings and is ideal for low-power applications. Since sub-sampling involves sampling below the Nyquist Rate, it leads to aliasing and increased noise at the baseband.

1. Shifted replicas of the signal and noise will overlap with each other.
2. A copy of all signal frequencies will be aliased to some lower frequency.

- One-Bit Data Processing

  Processing of 1-bit data allows for maximum power savings, and is thus very desirable for low-power applications [20]. In addition, since 1-bit quantization is a highly nonlinear operation, the preceding analog components need not be linear, and no gain control is needed. Nonlinear amplifiers track and hold circuits and samplers can be designed to use considerably less power than their linear counterparts. However, 1-bit quantization has three major negative effects on the receiver performance as follows.

  1. The received sinusoidal waveforms and noise are corrupted into rectangular waveforms.
  2. Quantization noise is introduced into the system, raising the noise floor.
  3. If signals are transmitted in adjacent channels, inter-modulation can cause these channels to fold on top of each other, making recovery impossible.

  The first two effects will degrade the performance of the proposed receiver. Intermodulation, however, is not an issue for space communications since only one communications channel exists at any given time.

- DFT-Based receiver architecture

  Instead using two correlators, as discussed earlier, to detect the binary FSK, a DFT is used. More specifically, a 16-point DFT is used to take advantage of the efficient radix-4 fast Fourier transform (FFT) algorithm. The DFT can be considered as a parallel bank of 16 separate correlators, each at a different frequency. However, by using the FFT algorithm, the outputs of all 16 correlators are computed using relatively few operations. The advantage of the DFT is that it allows the receiver to handle large frequency offsets without the need for down-converting the signal to exactly 0 Hz. This coupled with the highly efficient implementations of FFT, lead to significant reduction in power. The DFT detector also allows simple frequency tracking, making it robust against rapidly changing Doppler.

Chapter 3

Symbol Recovery

The symbol recovery circuit is an essential part of any digital receiver synchronizing the receiver clock with that of the transmitter. An error in the phase or frequency of the data clock can lead to a bit errors and needs to be corrected.

The Symbol recovery circuit extracts the clock information from the input modulated signal. This circuit often contributes to a large amount of power. This is because it needs to generate an internal clock and track the received signals clock. Most often these circuits are analog in nature and need large amounts of currents to drive the systems. The Phase Locked Loop or the PLL is the most commonly used Symbol recovery circuit, due to its relative simple design and easy tunability. In this chapter, we discuss the various types of the phase locked loops and their working.

3.1 Phase Locked Loop (PLL)

The phase-locked loop (PLL)[21, 23, 25, 24] causes a particular system to track another one. It synchronizes an output signal (generated by an oscillator) with a reference or input signal in frequency as well as in phase. The PLL is a control mechanism which acts on the oscillator when phase error builds up, and reduces the error. From a control system stand point the phases of the input and output signals are locked and hence the term Phase-Locked Loop. Figure 3.1 shows the basic structure of a phase locked loop. The PLL consists of three basic functional units:

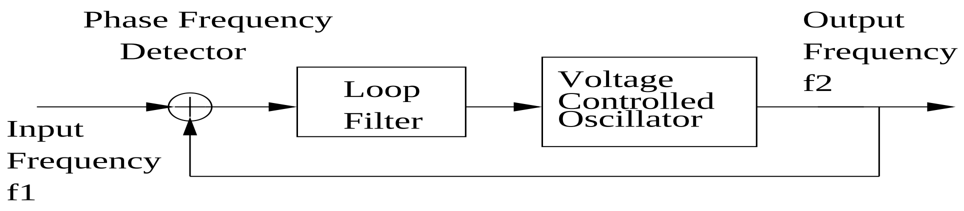

Figure 3.1: Phase Locked Loop

1. Voltage-controlled oscillator (VCO)

   The voltage controlled oscillator is a tunable frequency oscillator whose frequency is a function of the input. The VCO is responsible for tracking the data by changing its frequency to match it with the data. The angular frequency ω_2 is given by

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

   Where ω_0 is the center frequency of the VCO and K_0 is the VCO gain in s^{-1} V^{-1}. The center frequency is typically set to be the carrier frequency or the frequency at which the oscillator should oscillate if there was no input to the system. This can be a problem with VCO's as a free running VCO can consume a large amount of power.

2. Phase detector (PD)/phase-frequency detector (PFD)

   The PD-also known as the phase comparator, compares the phase of the output signal with the reference signal and develops an output signal u_d(t) which is proportional to the phase error θ_e,

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

   K_d represents the gain of the phase detector. The phase detector acts as a phase error amplifier which is used to control the variation of the VCO frequency.

3. Loop filter

   The output signal of the phase detector u_d(t) consists of a dc and a superimposed ac component. The latter is undesired, since it will lead to VCO frequency changing with the ac signal. The loop filter cancels this ac component by averaging the value over a period of time. Most often the loop filter is a low-pass filter of the first order. The filter also helps to stabilize the VCO and avoid rapid changes due to random noise at the input of the system.

Most PLLs are designed with their application in mind and hence the implementation of these functional units may differ considerably. However the basic function of the PLL still remains the same; to achieve synchronization in phase and frequency between the reference and the output clocks.

3.1.1 Operation of a PLL

Let us assume the frequency of the input signal is equal to the center frequency of the VCO. In this case the phase error is zero and the output of the loop filter also remains zero. The PLL is said to be in lock and remains so till a change occurs in the input.

If the frequency of the input signal is changed suddenly by a small amount, then the phase of the input starts leading the phase of the output signal. This phase error is built up and increases with time. The phase detector develops a signal which also increases with time. The output of the loop filter increases with time, leading to an increase in VCO frequency. With time the VCO frequency matches the input frequency. Depending on the type of loop filter, the final phase error will reduce to zero or to a finite value.

Depending on the various methods of lock and unlock PLLs can be classified into three broad categories which are discussed in the next subsections.

3.2 Linear PLL

The Linear PLL consists of a four-quadrant multiplier as the Phase detector and an analog filter and VCO. In most cases the input signal is a sine wave of frequency f_1 and the output is a symmetrical square wave of frequency f_2 which equals f_1 under lock condition.

Since the phase detector is an analog multiplier it contains a large number of frequency terms; the first of these is the DC term which is proportional to the phase error θ_e between the two signals; apart from these there are signals at harmonics of the fundamental at $2\omega_1$, $4\omega_1$, ... These components are unwanted as they do not represent the phase error, and need to be removed by the loop filter. Due to this the loop filter is a low-pass filter. The order of the low-pass filter determines the order of the PLL.

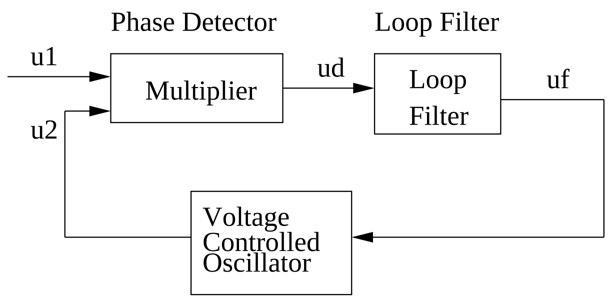

Figure 3.2: Linear Phase Locked Loop

In order to characterize the PLL, the transfer function of the output phase and the input phase namely, $H(s) = \frac{\Theta_2(s)}{\Theta_1(s)}$ needs to be calculated. It can be shown that the general form of the PLL's transfer function for a first order loop filter is given as [21, 23, 24, 25],

$$H(s) = \frac{2\eta\omega_n s+\omega_n^2}{s^2+2s\eta\omega_n+\omega_n^2}$$

Where η and ω_n represent the damping factor and natural frequency respectively. The PLL behaves as a low-pass filter for frequency values less than ω_n. As long as the modulation of the reference signal lies between 0 and ω_n the PLL will be able to track the signal. The damping factor η has an important role on the performance of the PLL. For $\eta = 1$ the system is critically damped. As we make η smaller the system becomes more and more oscillatory, with the overshoot increasing with decreasing η. For most practical systems a value of $\eta=\frac{1}{\sqrt{2}}$ is chosen, which represents a second order Butterworth filter. As we increase η the response of the system becomes more and more sluggish. Three conditions are necessary for the PLL to stay in the locked condition

1. The angular frequency of the reference signal must be within the hold range $\Delta\omega_H$

2. The maximum frequency step applied to the reference input to the PLL must be less than the pull out range $\Delta\omega_{PO}$

3. The rate of change of the reference frequency must be less than ω_n^2

The various key parameters of the PLL are summarized as follows:

1. Hold range $\Delta\omega_H$

   This is the frequency range in which a LPLL can statically maintain phase tracking. A PLL is conditionally stable only within the range. It is given by $\Delta\omega_H = K_0 K_d F(0)$.

2. Pull-out range $\Delta\omega_{PO}$

   This is the dynamic limit for stable operation of a PLL. If tracking is lost within this range, a LPLL will be $\Delta\omega_{PO} = 1.8\omega_n(\eta + 1)$ able to lock-again, but may be slow if it is a pull-in process.

3. Pull-in range $\Delta\omega_P$

   This is the range within which the LPLL will always become locked, but it can be a very slow process.It is given by, $\Delta\omega_P = \frac{4\sqrt{2}}{\pi}\sqrt{\eta\omega_n K_0 K_d}$.

4. Lock range $\Delta\omega_L$

   This is the frequency range within which a PLL locks within one single beat note between reference frequency and output frequency. Normally the operating frequency range is restricted to the lock range. It is given by $\Delta\omega_L = 2\eta\omega_n$.

In general, $\Delta\omega_L < \Delta\omega_{PO} < \Delta\omega_P < \Delta\omega_H$

The LPLL is an analog circuit and designing it in deep sub-micron technologies is often very hard. This is because of the large number of parasitics and leakage currents which need to be minimized. The VCO center frequency can change by a large amount due to these effects. The analog multiplier which is used as a phase detector suffers from a low pull-in range and issues like non-linearity and noise. Due to this these PLLs are not used in most modern day on-chip applications.

3.3 Discrete PLL

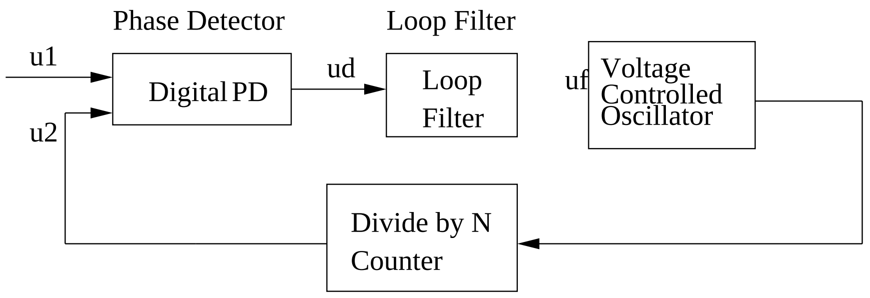

Figure 3.3: Discrete Phase Locked Loop

The Discrete PLL or the DPLL is a hybrid system consisting of both analog and digital components (Figure 3.3). The primary difference in the DPLL when compared to the LPLL lies in the design of the phase-detector. The analog multiplier used in LPLL's as phase detector provides limited pull-in range and is prone to problems typical in design of mixers, namely noise and linearity. There are three important kinds of phase detectors used in a DPLL:

1. XOR gate

The XOR gate phase detector is the digital equivalent of the analog multiplier. Since the two inputs to the phase detector are square waves the XOR gate multiplies the two signals giving an output of one when the signals are different and an output of zero when the signals are identical. The XOR gate is a linear function of the phase difference unlike the analog multiplier which is proportional to the sine of the phase error. However, like its analog equivalent it suffers from limited pull-in range and can only perform phase tracking when the phase error is confined to the range $-\frac{\pi}{2} < \theta_e < \frac{\pi}{2}$. The performance of this detector is also severely affected by asymmetries in the input waveform and hence is not very widely used. The phase detector gain if this detector when the logic levels are V_{max} and V_{min} is given by,

$$K_d = \frac{V_{max} - V_{min}}{\pi}$$

2. J-K Flip Flop

The problem with asymmetric inputs is taken care of by using a JK flip flop with edge triggered J and K inputs. The inputs of the flip-flop are connected to the two input signals and are edge triggered. A positive edge on the J input turns the output to a high state while a positive edge on K resets the output to zero. This phase detector has zero phase error when the input and output are the inverse of each other. The JK flip flop detector can handle a phase error in the range $-\pi < \theta_e < \pi$. However it suffers from limited pull-in range as the output of the detector depends only on the phase difference and not on the frequency difference between the two inputs. The phase detector gain if this detector when the logic levels are V_{max} and V_{min} is given by,

$$K_d = \frac{V_{max} - V_{min}}{2\pi}$$

3. The Phase-Frequency detector (PFD)

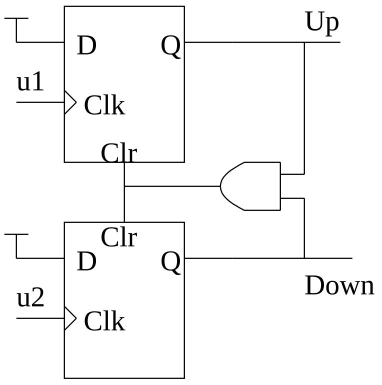

Figure 3.4: Phase Frequency detector

The Phase-Frequency detector is the most widely used phase detector. The PFD can sense both variation in phase and in frequency and hence offers virtually unlimited pull-in range. The PFD is shown in figure 3.4, consists of two D flip flops clocked by the two phase detector input signals. The flip-flops are connected to the supply rail in their inputs and are reset when both the outputs are high.

When the input lags behind the output the output of the PFD is proportional to the duration of this phase error. The PFD can handle a much large phase error in the range $-2\pi < \theta_e < 2\pi$. The phase detector gain if this detector when the logic levels are V_{max} and V_{min} is given by,

$$K_d = \frac{V_{max} - V_{min}}{4\pi}$$

The main strength of the PFD lies in its ability to distinguish the frequency differences. The duration of the Up and down pulses at the output of the PFDs give a very good measure of the frequency differences when the system is not locked. If the Up pulse remains high for much longer than the Down pulse, it means that the input frequency is higher than the output. The difference in this duration is proportional to the ratio of the two frequencies. This property makes it possible to offer unlimited pull-in range and make sure that the PLL locks under any situation.

3.4 All-Digital PLL

Unlike the other PLLs discussed the ADPLL is made up completely of digital parts. Not only are the blocks digital, even the signals involved in the ADPLL are digital. The ADPLL doesn't suffer from the issues of analog circuit design, like mismatch, non-linearity and effect of parasitics. It also consumes considerably lower power than a conventional LPLL. However, the ADPLL often needs a clock which is at a much higher rate than the inputs of the ADPLL and can only be used in situations where such high-rates are available. Since all parts of the ADPLL are digital, it can be built on-chip and the cost is much lower when compared to the other PLLs. The mathematical treatment of ADPLLs is usually more complicated than the others as they are non-linear systems and the dynamics of LPLL can often not be applied. The main components of the ADPLL are the same as the LPLL, namely, phase detector, loop filter and voltage-controlled oscillator. However, there is a large variety of designs in each sub-block depending on applications.

1. All-digital phase detectors

   As seen in DPLL, there are three main types of phase detectors- XOR, JKFF and the PFD. The ADPLL's phase detectors are extensions of these and can be classified as:

- FF-counter phase detectors

The FF phase detectors are extensions of the JKFF type PD with a counter added to keep track of the phase error. The Flip Flop phase detector is shown in figure 3.5 The reference signal and the output of the VCO are both binary

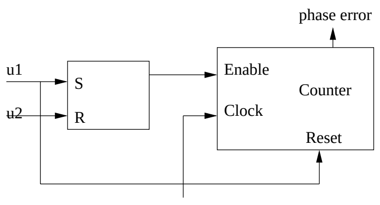

Figure 3.5: Flip Flop Phase Detector

valued signals, and are used to set or reset an edge triggered RS flip-flop. The time period for which the output of the flip-flop is high is proportional to the phase error. This is counted using a counter running at the high frequency clock and gives a number N θ_e, the phase error.

- Nyquist rate phase detector (NRPD)

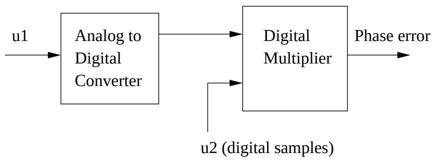

Figure 3.6: Nyquist Rate Phase Detector

As the name suggests, Nyquist rate phase detectors (shown in figure 3.6)are used when the sampling clock frequency is higher than the Nyquist rate of the signal. The Nyquist theorem states that in order to reconstruct a signal, the signal needs to be sampled at a frequency at least twice the highest frequency component in it.

The input to the phase detector is an analog signal which is sampled at a high frequency and multiplied digitally with the output of the VCO. The resulting signal is then averaged to get the average phase error, θ_e

- Zero-crossing phase detector

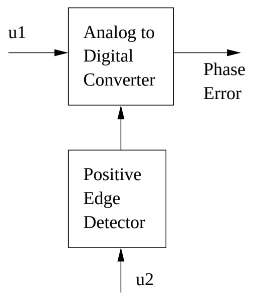

Figure 3.7: Zero Crossing Phase Detector

The zero-crossing phase detector uses the generated clock to sample the input data. The value of the sampled data converted to a digital output using an ADC. This value is latched till the next output clock edge and represents the phase error, θ_e

- Hilbert transform phase detector

The Hilbert transform phase detector extracts the phase error by using trigonometric computations. The DCO used in this kind of phase detector produces two signals and in-phase signal $I = \cos(\omega_0 t)$ and a quadrature signal $Q = \sin(\omega_0 t)$. If the input signal is a digital signal of the form

$$u_1(t) = \cos(\omega_0 t + \theta_e)$$

then by trigonometric operations,

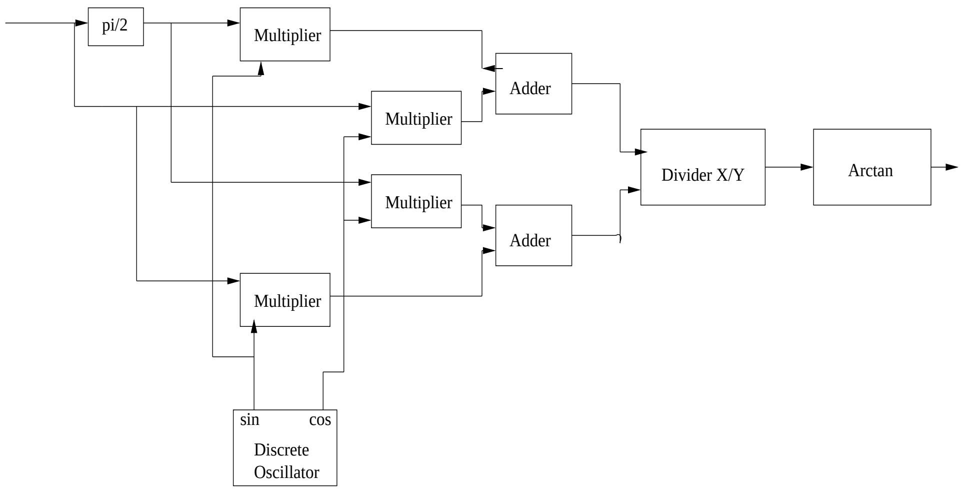

Figure 3.8: Hilbert Transform Phase Detector

$$\begin{aligned}\cos\theta_e &= Iu_1 + Qu_1^{'} \\ \sin\theta_e &= Iu_1^{'} - Qu_1\end{aligned}$$

Where $u^{'}$ represents the Hilbert transform of the signal. The two signals generated are divided to get the tangent and an inverse tangent is performed to get the value of the phase error. Due to the computational complexity, this kind of phase detector is used in software driven ADPLLs.

2. All-digital loop filters

Unlike the LPLL and DPLL counterparts the loop filters used in an ADPLL are made completely by digital components. There are a large number of variations in design of these filters. However, depending on the kind of phase-detector used we can divide these into three types:

- UP-DOWN counter

  The up-down type of loop filter is primarily used with a phase detector which generates two signals up or down, such as the PFD. It can however be easily adapted to be used with the JKFF and XOR type phase detectors. The up-down counter is a counter till N which increments the value of N if up is high and decrements the value of N if down is high. Over an interval of time the up-down counter essentially serves as a accumulator or discrete integrator, with a transfer function roughly given by,

$$H(s) = \frac{1}{sT_i}$$

- K counter

  The K counter is one of the most widely used loop filter for ADPLL designs. It is primarily used with phase detectors like the PFD, which generate two signals. The K counter consists of two separate counters, one counting when up is high and the other counting when down is high. Both these counter are modulo K. When the value of any of the counters exceeds K/2, a corresponding signal is sent to the VCO.

- N-before-M counter

  This type of loop filter is also used with the PFD and other phase detectors producing two signals up and down. the N-before-M filter uses two counters one counting down to M and the other counting down to N, such that M¿N always. There are two divide-by-N counters, one counting the up signal and the other the down signal. The M counter counts both up and down pulses. The N counters generate a high only if the M counter has not counted until M. In general, If there are N pulses of either Up or down in a total of M pulses (up or down) then the output of this filter goes high.

- Digital FIR/IIR filters

  Digital loop filters are equivalent to their analog counterparts and in many cases are derived from them using bilinear transformations and other techniques. Sometimes IIR filter techniques are used to design filters which act as VCOs by making the filter oscillate at its natural frequency[].These type of filters are used when we are trying to convert the LPLL into its corresponding digital equivalent. These types of filters are also commonly used with the Nyquist and the Hilbert transform phase detectors.

3. Digital VCOs or Numerically controlled oscillators

   The NCOs or numerically controlled oscillators are essentially frequency synthesizers which use a high frequency clock and produce a lower frequency one depending on the inputs to the filter. In this sense these VCOs are fundamentally different than there analog counterparts. The Analog VCOs are typically tunable LC oscillators and generate the frequency of interest. The NCOs consume much less power and current when compared to their analog counterparts, however they need a high frequency clock in order for them to function. A large variety of DCOs exist and their design is application dependent. Following are a few types of DCOs used:

   - Divide by N DCO

     This is the simplest kind of DCO which is essentially a Divide by N counter, generating a lower bit-rate clock from a high-frequency clock. The value of N is controlled by the output of the loop-filter.

- Waveform synthesizer DCO

  In this type of DCO the sine/cosine waveforms of the given frequency are generated using a look-up table stored in memory. The frequency of these signals is changed depending on a control signal. Due to the look-up table, the resolution of the generated signal depends on the frequency being generated. Lower frequencies have higher resolution than the higher frequencies. Due to the computational complexity of such DCOs they are primarily used in software/DSP based PLLs.

Chapter 4

System Level Design

4.1 Introduction

In the last two chapters, we discussed various techniques used in the design of space receivers and symbol recovery techniques circuits. In this chapter, the design of the DDPSK space receiver targeted for a planetary landing vehicles is discussed. The specifications for this receiver are similar to the receiver constraints discussed in Section 2.4 in chapter 2. Summarizing, the receiver needs to operate at a carrier frequency of 431.7MHz, under a Doppler rate of 10 kHz/s which can vary as much as 10Hz/s/s. A set of three different bit rates need to be supported including 10 Kbps, 1Kbps and 100bps.

The receiver [26, 27, 28] uses a novel modulation scheme called Double differential Phase Shift Keying (DDPSK) to counter Doppler shifts. The DDPSK receiver with 1-bit A/D is shown in Figure 4.1. The design of the baseband is such that it can support two alternative front-ends namely, sub-sampling front-end and a one-stage low IF down-converter. Depending on the front-end the signal to the A/D can have two different frequencies, the sampling frequency (for the sub-sampling case) or the low-If frequency. The current designs are meant for the low-IF front-end and can be modified easily for the other receiver. The input modulated signal is assumed to be down-converted to a low-IF of 1MHz and sampled by the sample and hold circuit at 4 MHz. The choice of the sampling rate leads to reduced complexity in the hardware.

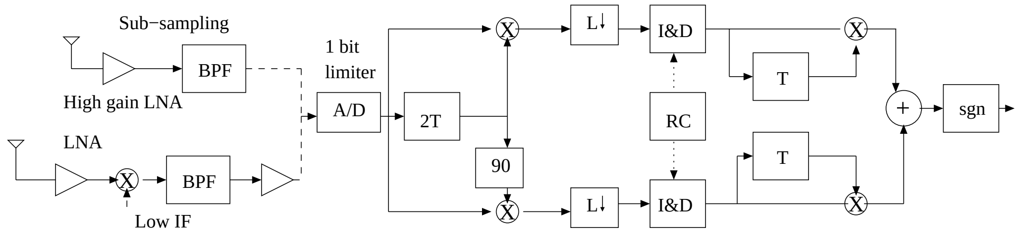

Figure 4.1: DDPSK receiver: block level model

In order to better understand the receiver implementation it was implemented and tested in MATLAB Simulink. The inputs to the system were generated through MATLAB scripts which modeled both the transmitter and the Doppler channel. The demodulator was tested for functionality and changes were done in it before moving on to the circuit level design and testing. An estimate of bit error rates due to Doppler and white noise was obtained.

4.2 DDPSK demodulator

A block diagram of the entire system is shown in figure 4.1 and its implementation in Simulink is shown in figure 4.2. The system can be divided into

1. Transmitter

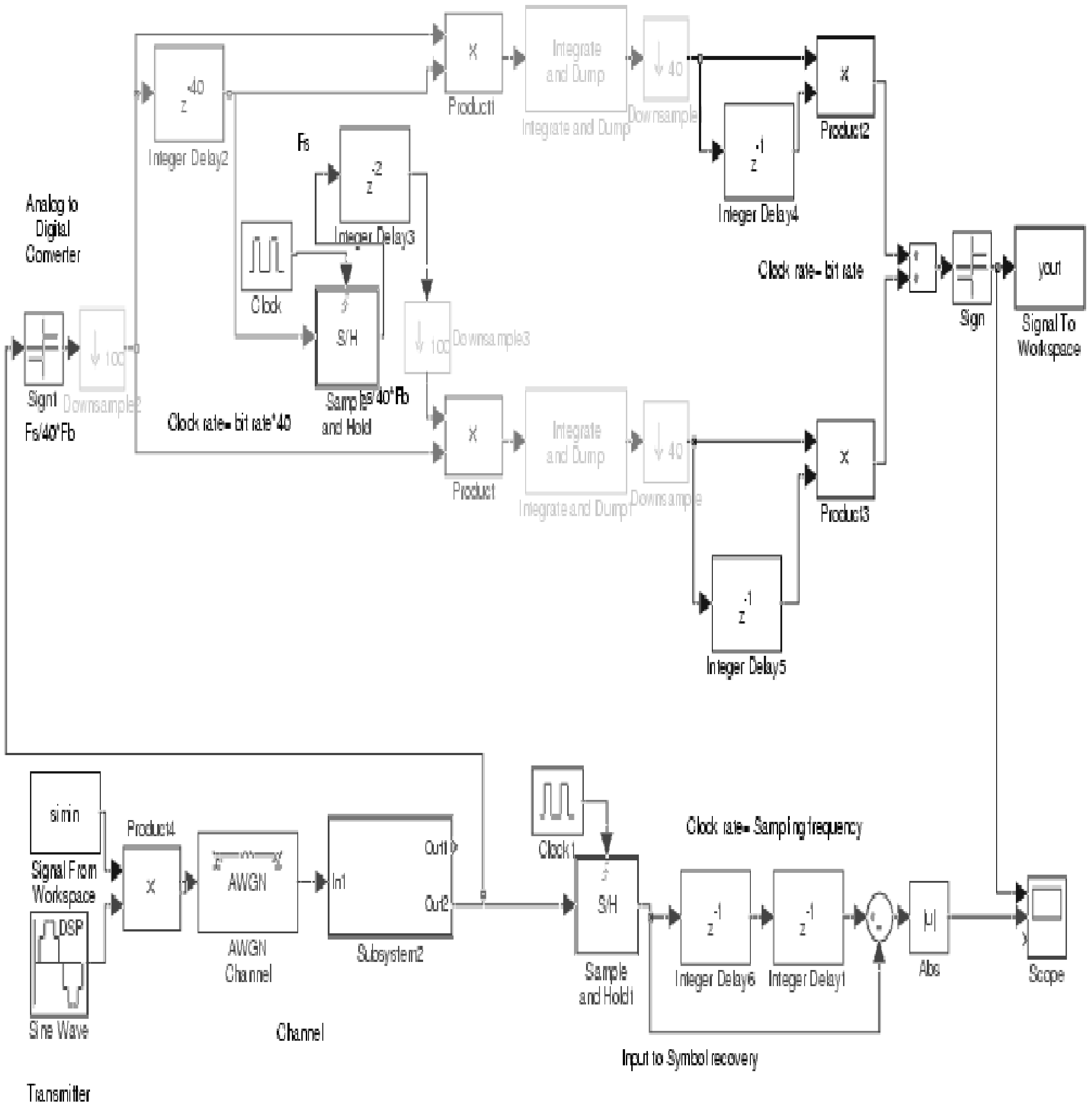

Figure 4.2: DDPSK receiver: Simulink implementation

The transmitter modulates the bit-stream using a 431.7 MHz carrier and uses DDPSK to encode the data. It then transmits the data over the channel. The transmitter was modeled by a MATLAB script which used the DDPSK equation discussed in section 2.2.2 in Chapter 2. The DDPSK bit stream is then multiplied with the carrier to obtain the transmitted signal.

2. Channel

   The deep space channel used is subject to both additive white Gaussian noise and to Doppler shifts due to the relative motion of the satellite around the planet There has been extensive study in this area and the treatment of the channel is very complex [1, 2, 3, 4, 5, 6]. The channel model used was a simplified version of the system and was implemented by using an AWGN channel and a Doppler spreading device. The characteristics of the channel have been described earlier in section 2.4.

   (a) Gaussian Noise

       AWGN was modeled using Simulink's communication toolbox, AWGN block. The value of SNR was changed depending on the specifications.

   (b) Doppler effects

       The channel in which the receiver operates is susceptible to Doppler shift variation due to the signal being transmitted from a moving satellite in orbit around the planet. However, since the satellite cannot change its velocity instantaneously, and has to do it in a progressive manner, the Doppler frequency changes only very slowly. The specifications given were to tolerate a Doppler shift of up to 10 kHz/s and a rate of Doppler shift up to 100 Hz/s/s. This was modeled using an S-function in MATLAB Simulink. The MATLAB code for the S-function is given in Appendix A.[29]

3. One-bit Analog to Digital converter

   The incoming analog signal is quantized into two levels at a sampling rate of 4 MHz. The A/D consists of a comparator and a sampling circuit. The choice of a 1-bit A/D was driven by the need for low-power design. The reduced power, results in additional performance degradation due to quantization noise. It can also lead to additional clock jitter errors in the system. The one bit A/D is modeled as signum block which is followed by a sample and hold circuit.

4. Bit-rate selector

   The receiver needs to support a wide range of bit-rates starting from 100kbps down to 100bps. In order to support such a large variation in bit-rates and yet maintain a relatively simple design, the clock rate of the receiver is controlled by using a bit-rate selector. This essentially is a register which samples data at the required bit-rate using the output of the clock recovery circuit. This block is modeled as a down-converter whose down conversion rate is controlled by the MATLAB code.

5. Demodulator

   The baseband demodulator consists of a two stage differential decoder in order to remove frequency shifts. The first stage implements an auto-corellator; it converts frequency error into phase error, and then the second stage eliminates the phase error.

The assumption is that the adjacent symbols are affected by the same frequency and phase shifts. As a result, the receiver utilizes non-coherent technique and does not require exact phase and frequency. The 1-bit A/D makes the design of the baseband completely digital as it replaces all the mixers by XNOR gates and the Integrate and Dump filters by up/down counters. The size of the accumulator is minimal, as the system works at a much lower rate. The demodulator implementation consists of the following sub-units

(a) Auto-corellator

The auto-corellator compares the signal with a delayed version of itself. It consists of a multiplier (XNOR gate for digital signals) with inputs as the signal and a delayed version of it. The auto-corellator demodulates one level of the DDPSK signal and thus removes the frequency variation effects from it. The output of the auto-corellator has only phase error effects in it.

(b) Integrate and dump filter

The integrate and dump filter is an accumulator(counter) which accumulates its value for a certain duration before resetting itself to zero and starting all over. This filter is a part of the auto-corellator and averages the value of the auto-corellator output over a bit-interval.

(c) Delay and multiply filter

The delay and multiply filter does the second stage of demodulation, using the output of the Integrate and Dump filter and doing a delay and multiply. This is the second stage of demodulation and is used to decode the signal completely.

(d) Decision device

The decision device adds the I and Q channels and determines makes a decision based on the sign of the signal. A negative value is considered a one while a positive value is considered a zero.

4.3 Results

After designing the system and describing it in MATLAB, the constraints on the system were analyzed and performance under various conditions ascertained. The design was tested for a large variety of inputs and these were used to estimate requirements for the clock recovery system. The receiver was intended to be designed for the highest possible bit-rate and through simulation runs, it was determined that the highest useful bit rate for a 1 MHz low-IF architecture, sampled at 4 MHz was 100 Kbps. This rate gave a very accurate determination of the clock.

As can be seen in figure 4.2, the delay elements are implemented in multiples of the bit-rate. This in turn implies that they could be implemented as a series of flip-flops working at the sampling rate. In order to support the lower bit rates, say 100 bps, with a sampling clock of 4 MHz, we would need 4 M/100= 40000 flip flops, which is not practical for the receiver. At 100 bps, the system does not need to work at such a high rate. Hence the system clock rate was reduced using the bit-rate selector. It was also found through simulation that the higher bit rate could be 40 times the current rate to produce minimum effect on performance and make the design of the clock recovery circuit simpler.

Chapter 5

Hardware Description

5.1 Introduction

The circuit was designed using Honeywell's Rad-Hard SOI technology which offers superior performance required for the space application while consuming very little power. The design was done using Verilog at the RTL level and synthesized using Synopsys. This proved to be easier than a full custom implementation as the blocks for the SOI library had already been defined and their layouts implemented. The designs were optimized for 0.80um TSMC CMOS process and then the resulting gate level design was transferred to Cadence and converted to the 0.35um Honeywell Rad-hard SOI process. This approach proved useful as the circuit was to be tested for a large number of clock cycles which was much easier to do on an RTL level than in a circuit level. The RTL design also gave an option of testing on a Xilinx FPGA board.

In this chapter we describe the clock recovery module along with its subsystems. The focus is on design methodologies and choices involved in the design. A brief description of the SOI technology follows.

5.2 Silicon on Insulator (SOI)

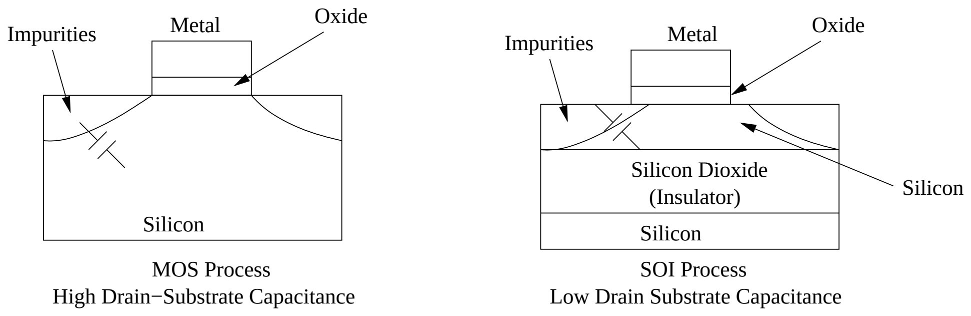

Figure 5.1: MOS and SOI processes

Silicon on Insulator technology [30] was first developed by IBM and is fast turning out to be one of the most promising for the development of low power and fast systems. Current fabrication techniques for such systems use the MOS (Metal oxide semiconductor) process which is shown in Figure 5.1.

In the MOS process, an oxide insulating layer is sandwiched between a thin layer of metal and a substrate of Silicon. Silicon as such is not a very good conductor of electricity and impurities need to be doped on a pure Silicon wafer in order to make it conduct better. When a low voltage is applied to the metal gate, there are not enough charge carriers on doped Silicon for it to start conducting. However as we raise the voltage, the charge carriers build up and the channel starts conducting when the applied voltage is greater than the threshold voltage. Hence, a MOS transistor acts as an on-off switch. The MOS and CMOS process have been responsible for the rapid growth of the digital industry. However, MOS transistor circuits are essentially capacitive in nature. There is capacitance between the gate and source, gate and drain and between drain and source and the substrate. With shrinking die sizes and smaller gate thickness, drain-substrate and source-substrate capacitances have become significant, while the current levels have kept on decreasing. This has led to slower speeds and higher power consumption, due to the need to charge and discharge the MOS capacitors. The SOI process reduces these bulk capacitances by decreasing the substrate area.

In the SOI process, shown in figure 5.1, an insulating layer (typically Silicon dioxide) is placed between the working junction of the transistor and the substrate Silicon. This effectively reduces the substrate area and hence the drain-substrate and source-substrate capacitances. Due to the reduced capacitances these circuits are much faster than circuits built using CMOS processes. SOI also is inherently radiation resistant (Rad-hard), as the substrate area is small, due to which the probability of cosmic radiation hitting the substrate is greatly reduced. These factors have led to a great interest in development of space applications using SOI technology. We now describe the design of the clock recovery circuit using Honeywell SOI technology.

5.3 Design

Among the wide varieties of phase locked loops described in Chapter 2, the all-digital version was chosen over the analog and discrete variants. This is because the emphasis was to design a circuit which would consume ultra-low power and be on-chip taking up minimal area. Since the signal was being over-sampled it was easier to generate lower clock rates using a frequency divider network. The highest bit-rate used was 100Kbps while the sampling clock was 4 MHz. This gave an accuracy of utmost 0.025 parts relative to the data rate, which was sufficient for the space application. The need for multiple bit-rates made it easier to have a programmable DCO and further motivated the design choices.

The receiver is synchronized by sending a pilot sequence of 01010101 whenever the bit-rate changes. The input is forced to have at least one transition every four cycles to stay in lock condition. Apart from the phase-frequency detector the circuit needs to have a mechanism to estimate the bit rate of the incoming signal and signal if there is a change in this rate. Hence, apart from the standard phase estimator there is a frequency estimator circuit to estimate these variations.

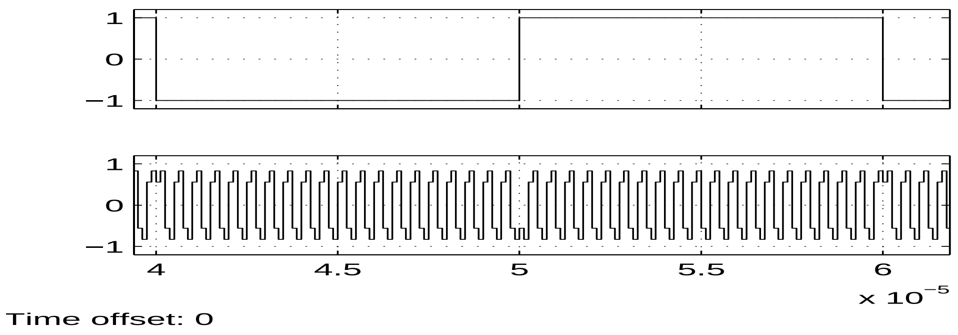

Figure 5.2: Input bits and Modulated Sequence

The input to the demodulator consists of a baseband low-IF signal at a carrier of 1 MHz and sampled at 4MHz. As shown in the figure 5.2 the input PSK signal changes in phase abruptly when data undergoes a transition. Hence, the signal inherently contains information regarding the clock edges; as long as there are sufficient data transitions. The input to the symbol recovery circuit consists of these edges representing clock pulses. These edges were achieved by XORing the input baseband signal with a delayed version of itself.

5.3.1 Block Level Design

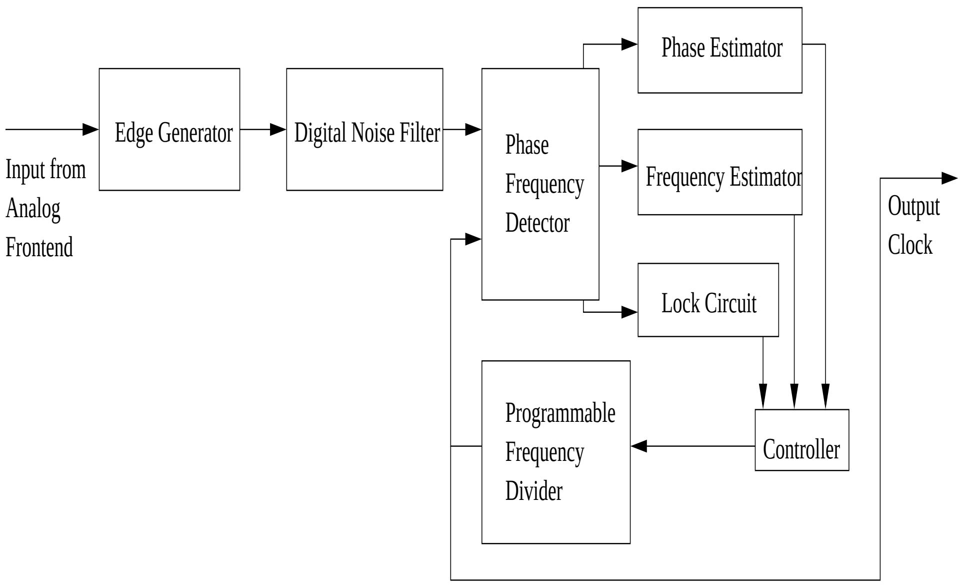

Figure 5.3: Block Diagram of Clock Recovery Circuit

Figure 5.3 shows the block level diagram of the symbol recovery circuit. The clock recovery module consists of the following components:

1. **Edge generator**

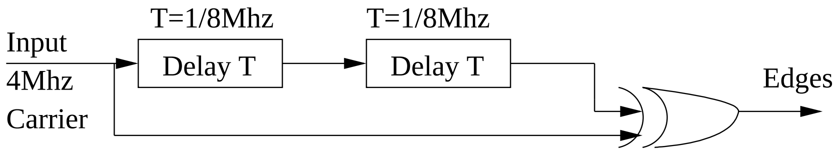

Figure 5.4: Edge Detector Circuit

For the PLL to lock accurately, a reasonable estimate of input clock is needed. The baseband input to the digital back-end of the receiver is a 1 MHz, low IF modulated signal sampled by 4MHz clock. Due to the nature of the DDPSK signal (Figure 5.2)

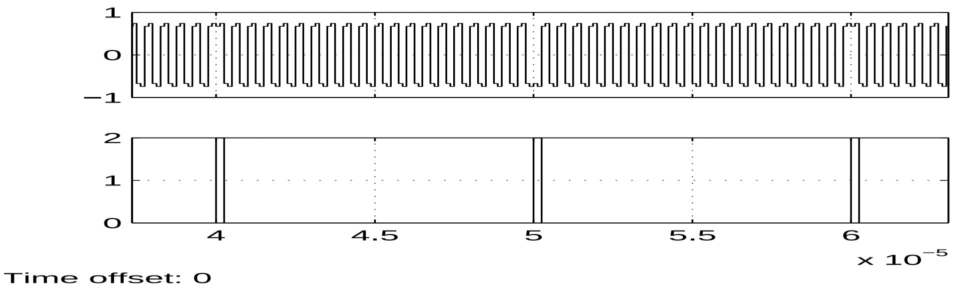

Figure 5.5: Received Sequence and Edge generator output

it is possible to extract the clock information from the received data. This is done by delaying the input signal by half a cycle and XORing it with itself. This results in pulse as shown in Figure 5.5. The delaying by half a clock cycle of the low-IF carrier is made easy due to over-sampling and amounts to using a two flip-flop shift register (Figure refedgedet).

2. **Digital Noise Filter**

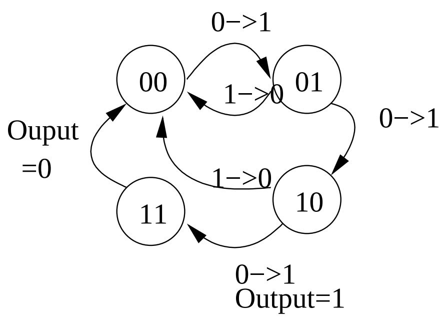

Figure 5.6: Digital Noise Filter: State Diagram

In a noise-free Doppler-free environment, the input to the system, which is a pilot sequence of 0101010101, would consists of the clock edges as shown in the figure 5.5. However due to AWGN and Doppler variations the input to the circuit consists of

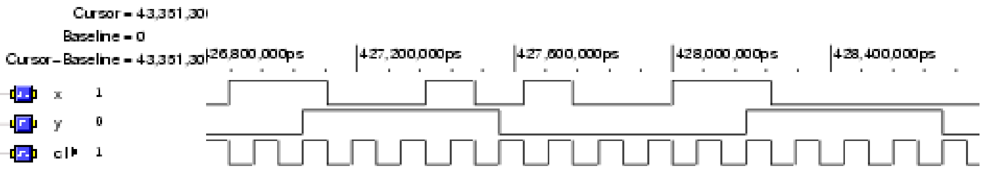

Figure 5.7: Received Sequence after Digital Filter - jitter is removed

a large number of jitter pulses along with the clock edges as shown in figure 5.7. Although these jitter pulses are of a very small duration, they can drastically affect the performance of the phase detector and need to be removed. The Digital Noise Filter removes the jitter from the input signal as shown in figure 5.7. The filter is a state machine where only signals of at least two clock durations to pass through and suppresses the other smaller duration pulses (Figure 5.6).

3. **Phase Frequency Detector**

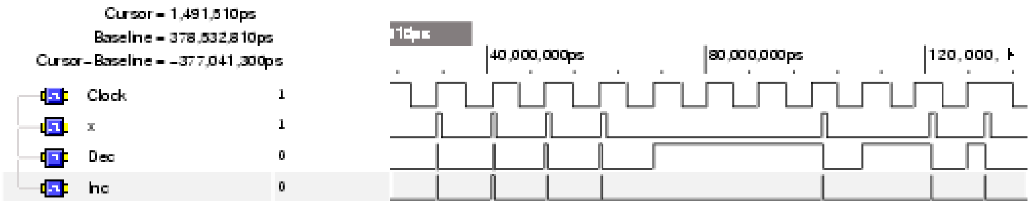

Figure 5.8: Phase Frequency detector: waveforms

The phase frequency detector acts as the phase and frequency estimator and generates relevant up and down signals. The PFD used is the standard PFD consisting of two D-flip flops with an asynchronous clear to reset the flip flops. This kind of detector has the advantage in detecting both the frequency and phase estimate and is essential for multi-bit rate systems.

4. **Phase estimator**

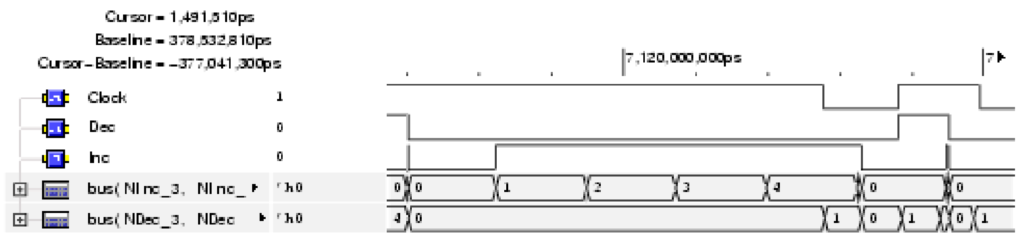

Figure 5.9: Phase Estimation

The phase estimator is used to determine the phase difference between the signals assuming that the frequency is the same. This is done by counting the number of Up and Down pulse and subtracting them, which gives a good estimate of the phase differences. However, the phase variation is used to change the phase only when there is no frequency error. This is because phase is a discrete integration of the frequency and phase error can also be caused by frequency variations. The phase estimator is reset to zero after the end of each clock cycle. This is done so that the phase estimates are relevant to the current cycle and do not represent a frequency shift.

5. **Frequency estimator**

The frequency estimator circuit determines whether the current frequency of the VCO needs to be increased or decreased. This is done by counting the number of UP or down pulses using the sampling clock of 4 MHz. The absolute value of the difference

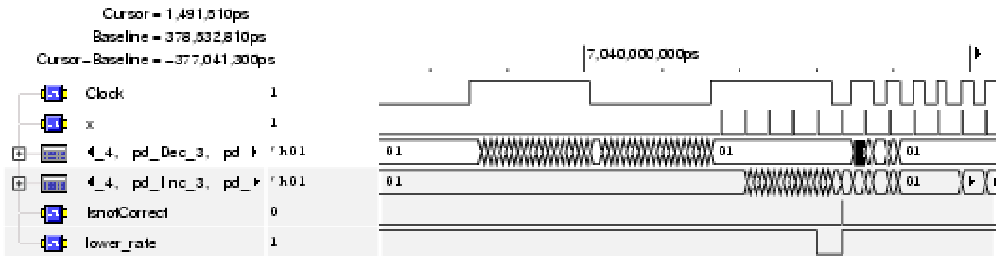

Figure 5.10: Frequency Estimation

gives an idea of the variation in the frequency of the incoming signal. The Up signal is high only if the input clock is ahead in phase/frequency with respect to the VCO clock. By counting the value of this signal and checking if it is greater than a threshold, it is possible to see if the frequency is above a certain margin. Similarly by counting the Down pulses it is equally possible to estimate if the input is lower in frequency than the output. A combination of these two and the threshold is then used to trigger a change in frequency through the control circuit.

6. **Programmable frequency divider**

   The frequency divider consists of a four constant frequency dividers of values of 10, 10 and 10 and 40 respectively, as shown in figure 5.11. Given that the clock is 4 MHz, this generates frequencies of 4 MHz, 400 kHz, 40 kHz and 4 kHz respectively. These frequencies are used to drive the rest of the baseband receiver. This measure helps drastically reduce power and area of the system. A programmable divide by 40 counter is used to down-convert this clock to the desired data rate. This arrangement ensures that the demodulator always functions at a rate 40 times the data rate. Due

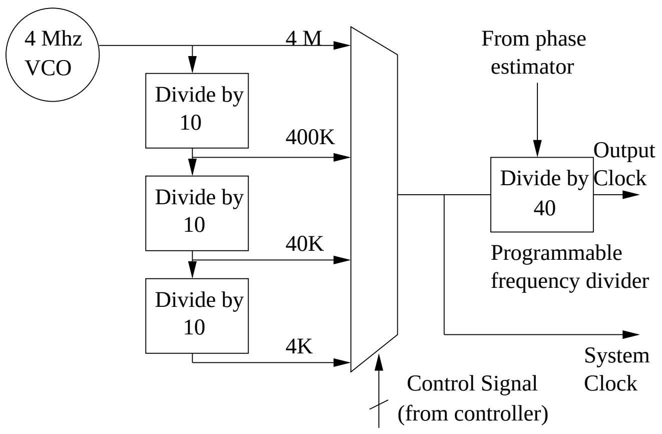

Figure 5.11: Block Diagram of frequency divider arrangement

to this it is possible to fix the demodulator architecture and hence reduce both power and area. In order to further reduce power consumption these dividers are switched on by the control circuit only when the relevant frequency is needed. An output multiplexer is used to select the relevant frequency.

7. **Lock Circuit**

When handling multiple bit rate signals, it is necessary to distinguish between sudden variations in frequency and a change in the frequency of the input signal due to a change in bit-rate. It is also important to make sure that the clock stays locked once the pilot sequence has been transmitted and data is being sent. This is because sequences like 0000011111... at 100Kbps are equivalent to 01... sequence at a bit-rate of 10Kbps.

In order to distinguish such sequences it is essential to have a lock detection circuit. The circuit along with the controller ensures that the output clock remains locked after the pilot signal has been sent. It does so by storing a sequence of phase error

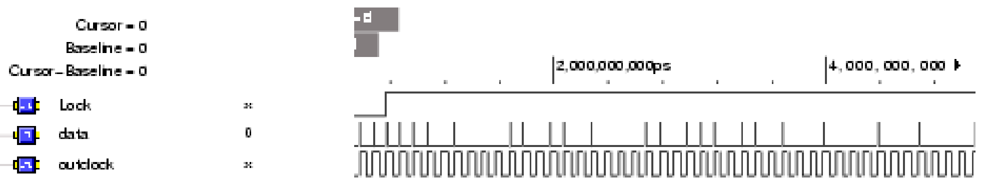

Figure 5.12: Capture, Locking and Resyncronization

values and repeating them as input to the discrete VCO. The lock detection circuit sends a lock signal to the controller once the phase error values are below a threshold error for a given number of clock cycles. Once locked, it bypasses the phase and frequency error values with the values stored. Every time there is a set of clock edges, the lock circuit resynchronizes to new phase and thus maintains the lock state. The lock circuit can be further sub-divided into:

(a) Shift Registers

This is a set of flip-flops which store the phase error values. When the system is in an unlocked state, this acts as a shift register, storing the phase error values through it. Once the signal gets locked, the shift register becomes a memory element looping the output back to itself. Since the amount of variation in the input sequence is relatively small, three flip flops were used to model the shift register function. There is a tradeoff between frequency detection and lock time, as the more the number of flip flops the better frequency variations can be resolved, but longer time to needed to lock the signal.

(b) Lock Controller

Once the system is locked the variation of phase errors is very small. It is possible to set a threshold value for the phase error below which we can say that the signal is locked. The Lock controller compares the output of the shift register and sets lock high when the value remains below a threshold for a fixed number of cycles. The number of cycles was chosen to be three and the threshold was chosen as one-sixteenth of the clock cycle. By varying these, the performance of the system can be fine tuned.

8. Controller

   The controller is responsible for the following:

   - Initializing the subsystems in the beginning and on a Reset
   - Increasing or decreasing the bit rate depending on the output of the frequency estimator
   - Bypassing the phase error when system is locked
   - When in lock, determines whether there is a need to unlock and recapture depending on the phase and frequency estimator values

5.3.2 Working

The system shown in figure 5.3 is an All-digital phase locked loop. The input to the system consists of the PSK pulses as shown in figure 5.2. The edge detector circuit extracts the sudden phase change information from the high frequency PSK signal. In the presence of no noise and Doppler, the BPSK signal changes phase abruptly whenever there is a transition in the data. This property can be used to extract information about the clock. The input to the receiver consists of a pilot sequence, 0101010101, whenever the bit rate changes. This gives enough time for the circuit to change to the new bit rate and lock to the phase. Sequences which have no transitions for more than four cycles are forbidden in the receiver. This is because a sequence like 1111100000 at 100Kbps is identical to sending a 10 at 10Kbps. Hence, sequences like 11111 or 00000 are not permitted in the receiver. When the system is in lock, there needs to be at least one data transition every five cycles to remain in a locked state.

The edge detector circuit uses the property of the BPSK signal, discussed above, to extract clock information. However, the circuit produces additional jitter pulses in the presence of noise and these are removed by the digital filter. The outputs of the digital filter are the edges of the data and hence represent clock intervals. The PFD uses these and the output clock to generate up and down pulses as shown in figure 5.8. These pulses are sent to a phase estimator, which counts the number of clock pulses of the next higher clock rate in one interval of the lower clock rate (If the current bit rate is 10Kbps, the clock used to measure phase is 400Kbps, the number of transitions in one cycle of 10Kbps are counted using the 400K clock), giving an accuracy of 1/40.

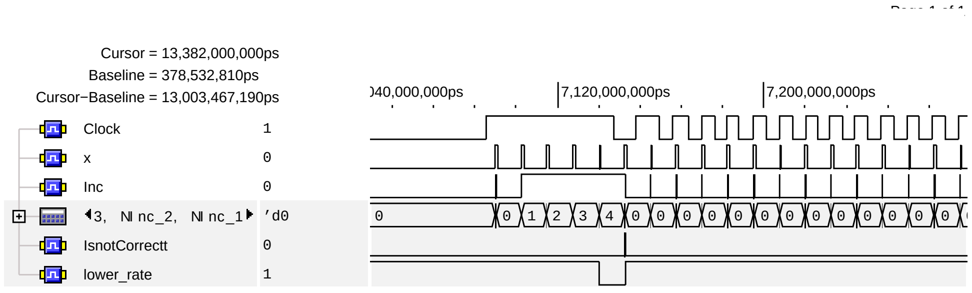

Figure 5.13: Changing to a higher clock rate

The frequency estimator circuit keeps track of the number of times there are edges when the Inc (or up) signal of the PFD is high and the duration for which the Dec (or down) signal are high. Figure 5.13 shows the way the higher bit rate signal is detected. Inc(or Up) signal of the PFD is high only when the input clock leads the output clock in phase or is higher in frequency relative to the output clock. If the signal leads in phase then the Inc signal would remain high just for one clock cycle before going back to zero. However, if it remains high for more than once cycle of the input, then it either means that there is some additional jitter in the system or the bit rate of the signal has changed. If five such transitions of the input occur when Inc is high, it is deduced that the clock rate has increased to the higher bit rate. This is a fair assumption as five noise pulses are highly unlikely to occur due to the noise filtering done by the digital filter.

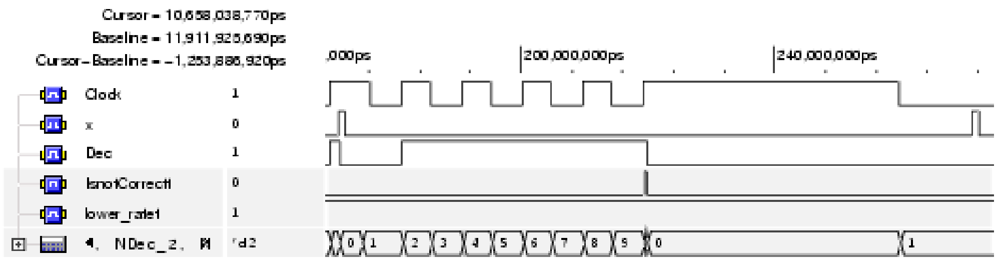

Figure 5.14: Changing to a lower clock rate

Figure 5.14 shows the way a lower bit rate signal is detected. If the Dec signal is high, it means that the input is lagging compared to the output, or the data rate is changing or data being sent does not have enough transitions. If the input is lagging relative to the output and the bit rate is the same, the Dec signal will go back to zero after one clock cycle.

The modulation scheme needs to have at least one transition every five cycles and hence if the Dec signal lasts for more than five cycles of the current clock (50 cycles in the higher clock), then the clock rate is decreased to the next lower rate.

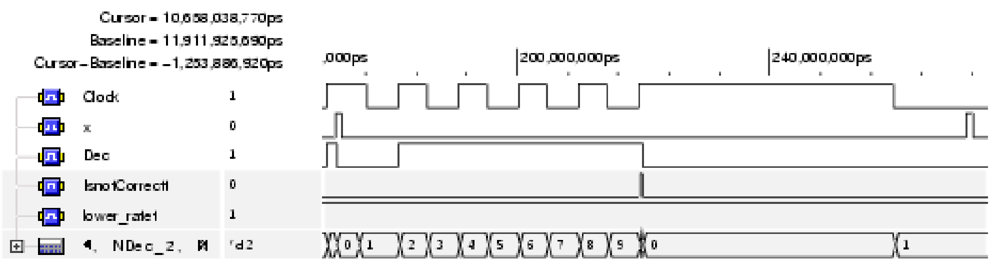

Figure 5.15: Lock mode - Inc and Dec are both below a threshold

Figure 5.15 shows the behavior of the lock circuit. The lock circuit keeps monitoring the value of the phase error and if it goes below a certain threshold and remains below it for three cycles, it sets the lock signal high. When phase lock is achieved the phase error between the input and output clocks is very small and this fact is used to achieve lock. When data is being transmitted, it is very hard to distinguish between a phase error and a lack of a transition. By keeping the circuit in lock and re-synchronizing the clock whenever a transition occurs, it is possible to keep the phase error to a minimum. As can be seen from the figure 5.15, there may be a large variation in the output of the phase estimator, due to lack of transition s but these do not change the clock rate. This is because the system is in lock mode and phase synchronization is maintained. It can also be seen from the figure 5.15, that a phase error is corrected as soon as two transitions occur in the input.

The frequency divider consists of a cascade of dividers which are connected to a multiplexer. Depending on the inputs to the controller and the lock, the controller sends the relevant selection to the multiplexer and changes the rates.

5.3.3 Power reduction

Since the main aim was to produce a very low-power design, a large number of architectural adjustments were made to reduce power of the system. From a circuits perspective, we know that the power consumption in any digital system is given by

$$P = \alpha C f V^2$$

where f is the frequency of the system, V is the supply voltage, α is the a proportion of the circuit undergoing transitions and C is the total load capacitance in the system. Since we are using an SOI technology for the design the capacitance in the system is already quite low. The supply voltage is fixed at 2V for the baseband system and cant be changed. Hence the two other factor namely, frequency and switching factor, had to be reduced to reduce power.

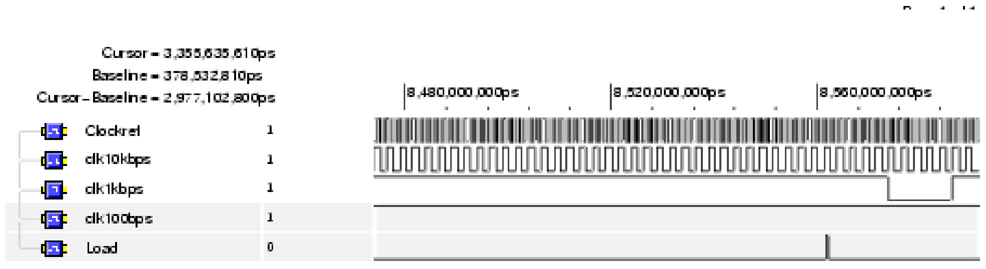

Figure 5.16: Power Reduction by clock gating - Lower clocks are off till frequency changes

The highest frequency available to the system is the sampling frequency 4 MHz. The data runs at much lower rates of 100Kbps and below. Hence, wherever possible the lowest clock in the system was used to run various components of the circuit. This made it possible to reduce the frequency. Extensive clock gating was done to make sure that unnecessary transitions do not occur. The clock divider circuit consumes a large amount of power as it is always switching at 50%. The clock divider was gated so that when a bit rate of 100Kbps was detected, the other bit rate clocks were turned of, this is shown in the figure 5.16.

Since the clocks are separated by a factor of 10(100K, 10K, 1K and 100), it was possible to use the higher clock to estimate the phase error of the lower clock to a high degree of accuracy. This led to decreased power consumption at the cost of reduced frequency resolution. It also reduced the capacitative load on the sampling clock and further helped to reduce power. Apart from the edge detection circuit, digital filter and the frequency estimator, which run at the sampling clock rate the rest of the system runs at a lower clock rate. The digital filter also reduces the noise in the system and reduces power consumption.

Another method used to reduce power was by using static logic to design the system. Dynamic logic is used when the system needs to be very fast. It consumes more power as it has to be pre-charged every cycle and hence the number of transitions is very high. Since power and stability were the most important criteria, the design was done wholly using static gates. This also made it easier to follow the Verilog-Synopsys path flow, as synthesizing dynamic logic is not possible in Synopsys.

5.4 Verification

5.4.1 Gate level verification

The clock recovery circuit was thoroughly tested using various varieties of input waveforms. Care was taken to ensure that sudden changes in bit rates were recognized and system retained phase lock. The design was done using Verilog RTL and then synthesized in Synopsys. Finally the synthesized Verilog code was ported to Cadence using the import option in Cadence tools. The design was then converted to SOI libraries by using some scripts. This is not necessarily the best design flow for the above circuit, but led to a much faster design and verification flow. Since the SOI libraries and the corresponding layout were already available, it was easier to use this flow instead of a full-custom design methodology.

The system was verified at all levels, at the design stage in Verilog, after synthesis using TSMC 0.80um libraries and then in the circuit level using Cadence and Spectre and 0.35um Honeywell SOI libraries. Care was taken to make sure that the results were replicated at all stages.

The test pattern followed was to ensure that the circuit would work for all possible bit rates, achieve phase lock at a given bit rate, stay in once the phase lock was achieved, and finally change to a new bit-rate when input data changed. The circuit was initially tested using no noise or Doppler effects, to make sure that the above steps were verified.

The initial series of tests involved working the PLL at the rate of 100Kbps and verifying if it achieves phase lock for the exact rate, namely 100Kbps, and then for nominal rates, like 99.9Kbps. Then AWGN alone was added to see if phase lock was still achieved and finally the Doppler induced signal was tested. These inputs were provided using the

Simulink model developed in the previous chapter. The same process was repeated for each bit-rate. At this stage no effort was made to change the bit-rates. These tests are shown in figures 5.17, 5.18, 5.19 and 5.20

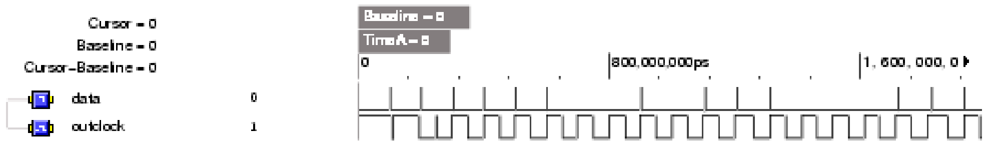

Figure 5.17: Phase recovery at 100Kbps

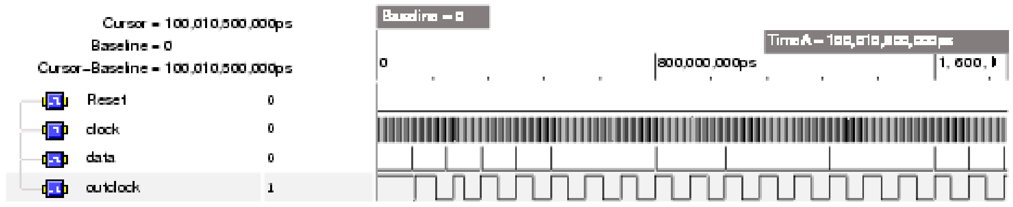
Cursor = 100,010,500,000ps
Baseline = 0
Cursor–Baseline = 100,010,500,000ps
Reset
clock
data
outclock
Baseline = 0
TimeA = 100,010,500,000ps
0
800,000,000ps
1, 600,

Figure 5.18: Phase recovery at 10Kbps

The next series of tests involved modifying the bit-rates and checking how the system responds. This was done by starting at a particular bit-rate, say 100Kbps, till the system locked and then changing the rate. This procedure was repeated until all possible combinations were exhausted. For example sequences like 100K, 10K, 1K, 100, 100K, 1K, 10K, 100, 1K, 100K... and so on, just to make sure that the PLL was sensitive to all such transitions. This process was verified with noise and without noise, with Doppler and without it. These tests are shown in figures 5.21, 5.22 and 5.23.

The final sequence of tests involved testing the lock conditions. The PLL was

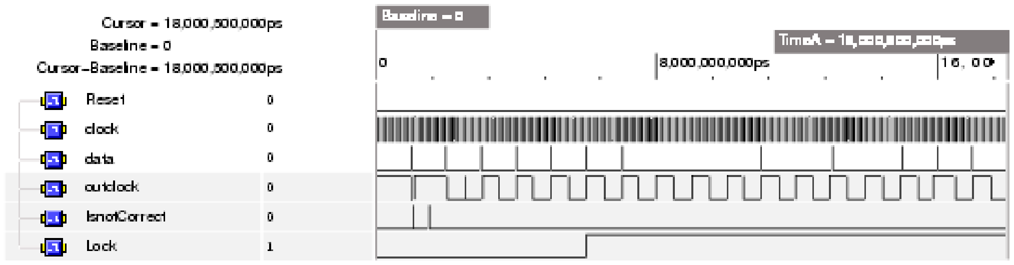

Figure 5.19: Phase recovery at 1Kbps

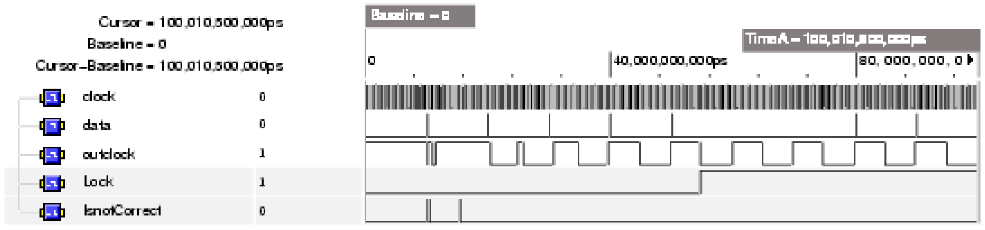

Figure 5.20: Phase recovery at 100bps

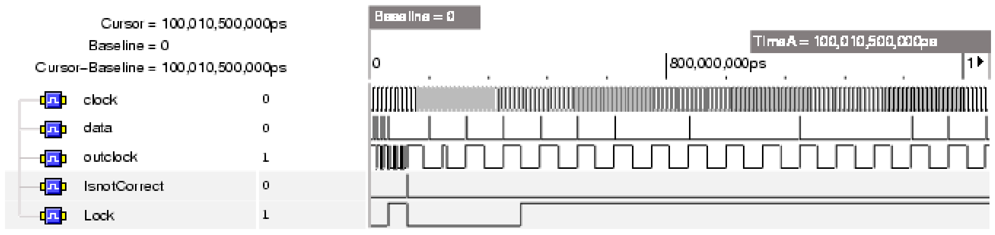
Cursor = 100,010,500,000ps
Baseline = 0
Cursor−Baseline = 100,010,500,000ps
Baseline = 0
TimeA = 100,010,500,000ps
0
800,000,000ps
1
clock 0
data 0
outclock 1
IsnotCorrect 0
Lock 1

Figure 5.21: Bit rate changes and Phase recovery 100Kbps - 10Kbps

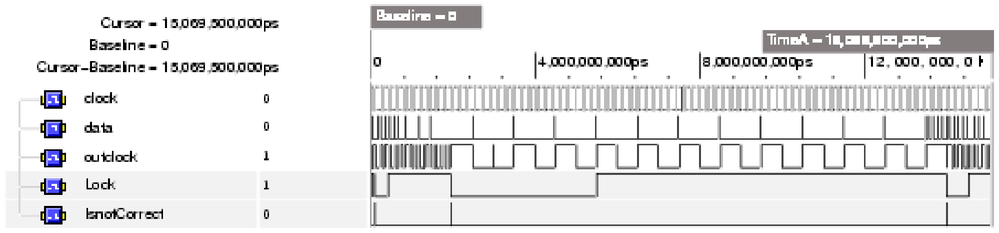

Figure 5.22: Bit rate changes and Phase recovery 100Kbps - 10Kbps - 1Kbps - 10Kbps

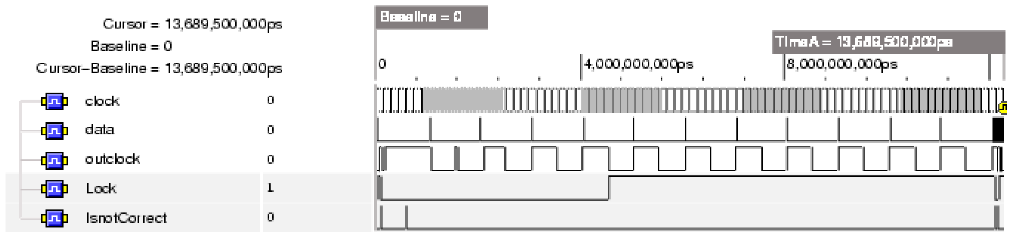

Figure 5.23: Bit rate changes and Phase recovery 100Kbps - 1Kbps - 100Kbps

made to achieve lock and then artificial phase errors were introduced to make the system out of sync. A random sequence of data was transmitted, making sure that there was at least one transition in four cycles, and response of the circuit verified to see if it maintained phase lock and minimized phase error. These tests were repeated at all frequencies and verified. Figure 5.24 shows an example of such a test.

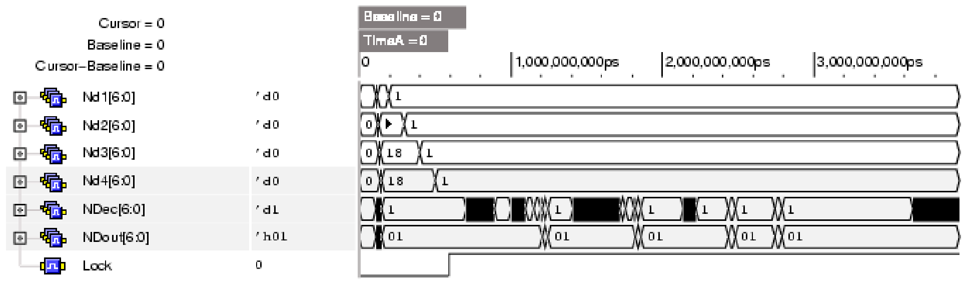

Figure 5.24: Lock

These tests were repeated for all frequencies first just in Verilog, then as a post-synthesis verification using the Synopsys 0.8um libraries and finally after porting it to SOI in Cadence, they were tested using Spectre. At all these stage it was made sure that the results were identical and correct. Since the circuit needed to be run for a large number of cycles in Spectre, the tests were more restrictive and involved just verifying the functionality.

5.4.2 Timing verification

Wwe need to make sure that there are no race conditions or setup violations in the circuit. The circuit was tested using TSMC 0.8um and then changed to 0.35um SOI which is a much faster logic family, and hence this verification is essential.

The main flip-flop used for design was the D Flip-Flop using C^{2}MOS latches. All other flip-flops like the DFFR and DFFS were based on the above flip-flop. In order to estimate the setup and hold times, the design was simulated by varying the D to clock time and measuring the clock to Q delays. The D-CLK delay at which 95% value of the minimum clock to Q delay was estimated to be the setup time. In a similar fashion, the hold time was estimated to be the D-clock time at which the clock to Q delay is the minimum, when coming from the right side of the clock edge.

D-CK	CK-Q	CK-Qb	D-Q	D-Qb
1n	395.4p	612p	1395p	1612p
400p	400p	616p	800p	1016p
300p	406p	623p	706p	923p
210p	442p	657p	652p	867p
200p	449p	666p	649p	866p
190p	460p	677p	650p	867p
180p	481p	697p	661p	877p
170p	533p	748p	703p	928p
160p	Meta-stable			

Table 5.1: Setup Time Analysis

160p	492p	720p	652p	880p
140p	386p	618p	526p	758p
120p	371p	601p	491p	721p
100p	366p	596p	466p	696p
0p	360p	590.4p	360p	590.4p
-200p	363p	592p	163p	392p
-1n	363p	592p	637p	408p

Table 5.2: Hold time analysis

The above analysis results are shown in table 5.1 and 5.2. From these table the values of setup and hold times for the D-Flip Flop are estimated. The setup time was found to be $t_{setup} = 280ps$ and the hold time was estimated to be $t_{hold} = 0$. This is valid as the flip-flops were chosen to be a master-slave which do not have race condition issues. The minimum clock to Q was found to be $t_{Clk-Q_{min}} = 363ps$ and the maximum clock to Q before meta-stability was found to be $t_{Clk-Q_{max}} = 533ps$. In order to meet setup time violations we need

$$t_{Clk} \geq t_{skew} + t_{logic-max} + t_{Clk-Q_{max}} + t_{setup}$$

The clock skew can be estimated from the hold time criteria,

$$t_{hold} + t_{skew} \leq t_{logic-min} + t_{Clk-Q_{min}}$$

Assuming worst case situation of zero logic delay and substituting the values of the delays, we get, $t_{skew} \leq 363ps$. The 4MHz VCO needs to have a skew less than this in order for the circuit to function correctly. The skew estimates of the VCO were well below the maximum skew requirements and the chances of hold-violation are minimal.

The setup violation estimates were made based on the worst case skew and logic delays. The logic delay estimates were derived from Synopsys design compiler. These are highly pessimistic since the SOI technology is much faster when compared to TSMC 0.80um. Substituting these values, the minimum clock rate needs to be $t_{Clk} \geq 363ps + 16.61ns + 533ps + 280ps \geq 17786ps \geq 17.786ns$. This represents a maximum operating clock frequency of 56.2 MHz. Since the sampling rate is 4 MHz, the circuit is well within the range of operation.

A summary of the symbol recovery circuit is given in Table 5.3. The circuit was further tested in SOI using Spectre and the SOI libraries provided by Honeywell. The results of the test are shown in Figures 5.26 and 5.27. The circuit layout was done in Cadence Virtuoso and is shown in Figure 5.25. The power values were estimated by simulating the extracted circuit and determining current levels with a Vdd of 2 Volts.

Input signal frequency, f_1	1 Mhz
Sampling frequency, f_s	4 MHz
Data rate, f_b	0.1-100 Kbps
Length of preamble, N_L	$\geq$ 4 bits
Power dissipation	155μA @ 2 V

Table 5.3: Timing circuit parameters

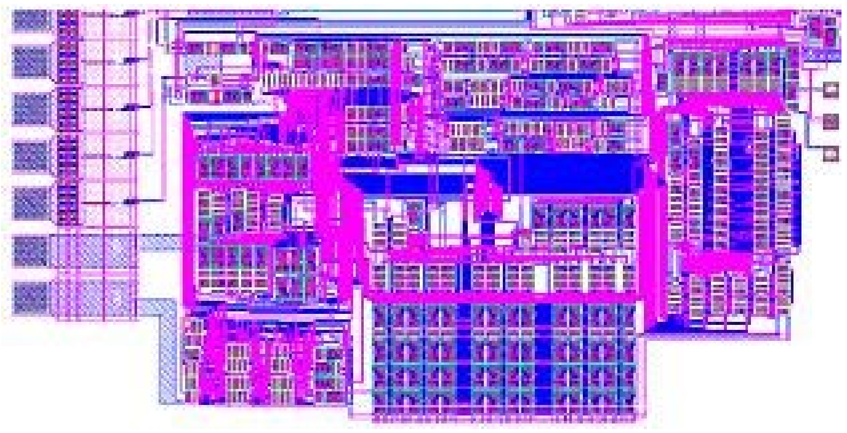

Figure 5.25: Layout of the circuit in SOI

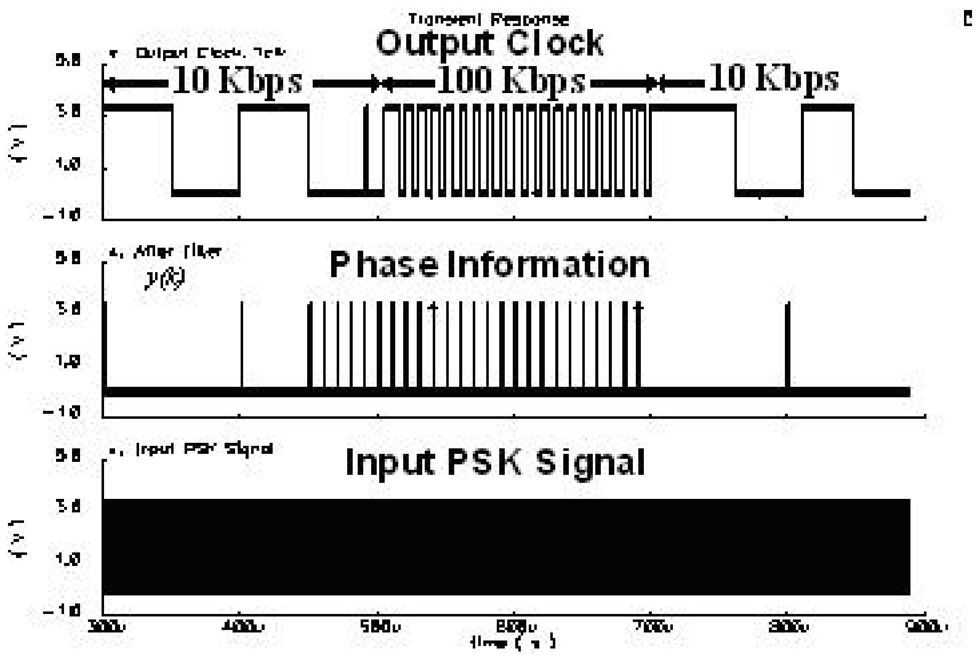

Figure 5.26: Test using Spectre

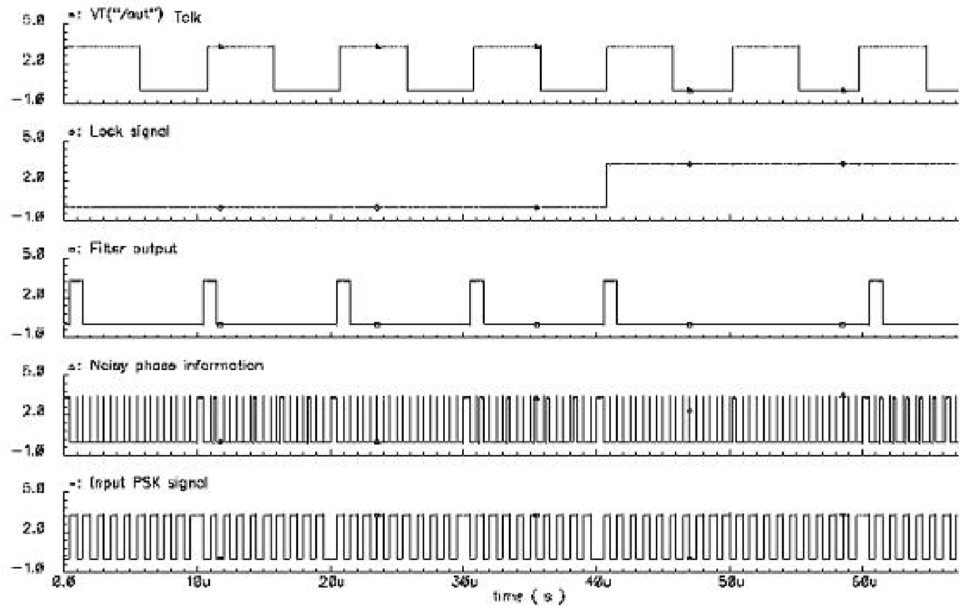

Figure 5.27: Test using Spectre

Chapter 6

Conclusion and Future Work

A design of the receiver for application to a planetary land rover was proposed. The receiver uses a Double Differential Phase Shift Keying scheme which provides inherent resistance to Doppler shifts. The receiver has a potential for very low power consumption while occupying a small area on chip. The receiver uses low-IF, 1 bit A/D in order to reduce power and make the circuit much simpler, while trading of performance.

The design of a symbol recovery circuit for the above receiver was proposed, designed and implemented. The circuit is capable of handling a variety of bit-rates while consuming minimal power. The circuit uses some novel methods to achieve phase and frequency lock. These methods are highly programmable and can be traded between lock time and frequency selectivity. The circuit uses a digital variant of the phase locked loop to reduce power consumption. The design uses the minimum possible clock rates in order to reduce power consumption. Clock gating is used wherever permissible to further reduce power consumption. The digital frequency divider is highly programmable and can be modified to handle a large variety of bit-rates.

The implementation of the circuit was done in Honeywells 0.35um, Rad hard SOI process. The power consumption of the circuit was found to be 310uA at a 2V Vdd.

The receiver trades of power for performance. In order to improve performance while keeping power levels the same, an error control code could be incorporated. The receiver can be integrated with a trellis coded modulation scheme to reduce the bit error rate. An alternate solution would be to add a Viterbi encoder-decoder to the system. It would also be possible to use a multiple symbol detection scheme to improve the basic performance of the scheme which could be further improved with error control codes. The signals could be transmitted with a non-NRZ system which will have the clock information irrespective of the data being sent. This could reduce the unnecessary overhead of sending synchronizing sequences along with the data.

Appendix A

Source Code

A.1 Verilog Source

```
module clockrecoverer(Reset,Clock,data,outclock);
input Clock;
input Reset;
input data;
output outclock;
wire datao,Change_rate,Bypass,Load;
simple S1(Clock,Reset,data,datao);
//A simple filter
sets_the_bypass B1(Clock,Reset,datao,Bypass);
//this runs at the ref clock rate, may have to change that
//this is because the clock divider uses bypass as an input
clockbyN C1(Reset,Clock,Load,lower_rate,clock);
//This is the programmable clock divider circuit
controls_the_count C2(~Bypass,Clock,clock,datao,Load/*Change_rate*/,
lower_rate/*l_rate*/,outclock);
//The set of counter which are the decision makers
endmodule

module clockbyN(Reset,Clockref,Load,lower_rate,Clockout);
//Implements the frequency synthesizer
input Reset;   // synchronous reset ( to reference clock 4Mhz??)
input Clockref; // Reference Clock
input Load;      // Load Control Signal ( synchronous to reference clock)
input lower_rate;
```

```
output Clockout; // The output clock
wire [1:0] whichrate; //Counter and the Value register
wire [1:0] cntrl_rate;
wire [2:0] Incr,Decr;
reg [2:0] code;
//control
counter_4 clockspeed(Load,Reset,lower_rate,whichrate);
//data
counter_10 clock10kbps(Clockref,~(|whichrate),clk10kbps);
counter_10 clock1kbps(clk10kbps,~whichrate[1],clk1kbps);
counter_10 clock100bps(clk1kbps,~(&(whichrate)),clk100bps);
multiplexor4_1 clockmux(Clockout,Clockref,clk10kbps,clk1kbps,clk100bps,
whichrate[1],whichrate[0]);
endmodule

module counter_10(Clock,Reset,DividebyN);
//Divides the clock by 10 and outputs it in divide by N
input Clock,Reset;
output DividebyN;
parameter N=3'd5;
reg [2:0] Count,Nreg;
reg DividebyN;
always @(posedge Clock or posedge Reset)
if (Reset)
     begin
         Count<=9'd0;
         DividebyN<=1;
     end
else
     begin
         if (Count==(N-1))
             begin
                 Count<=9'd0;
                 DividebyN<=~DividebyN;
             end
         else
             Count<=Count+1;
     end
endmodule

module counter_4(Clock,Reset,lower_rate,Count);
//2-bit counter , selects from the multiple clocks
input Clock,Reset,lower_rate;
output [1:0] Count;
```

```
reg [1:0] Count;
//Simple 2-bit Counter with increment/decrement
//if lower_rate is high then increment
// else decrement
always @(posedge Clock or posedge Reset)
if (Reset)
     Count<=2'd0;
else if (lower_rate)
     Count<=Count+1;
else Count<=Count-1;
endmodule

module counter_80(Clock,Reset,NInc,NDec,Nreg,x,DividebyN,Clear);
//Divide by N counter(N is 80 here)
//This N is the basic downconversion from 8Mhz down to the highest
//clock rate
input Clock,Reset,x;
//Count to 80
input [6:0] NInc,NDec,Nreg;
output DividebyN,Clear;
//Clock divider by  N
// If Inc is high=> step by 2
// If Dec is High=> step by 0
// If both are high/low => step by 1
//An issue: have to figure out whether to change N or the count
//N change=>change in frequency
//count change=> change in Phase
reg [9:0] Count,Count1;
reg DividebyN,Clear,change;
wire Inc_Dec,Dec_Inc;
assign Inc_Dec=(NInc>NDec)?1:0;
assign Dec_Inc=(NDec>NInc)?1:0;
always @(posedge Clock or posedge Reset)
if (Reset)
     begin
         Count<=9'd0;
         DividebyN<=1'b1;
     end
else
     begin
         if (Count==(Nreg))
             begin
                 Clear=1;
                 if (Inc_Dec)
```

```
                     begin
                         Count<=NInc;
                         DividebyN<=~DividebyN;
                     end
                 else if (Dec_Inc)
                     begin
                         Count<=Nreg-NDec;
                         DividebyN<=DividebyN;
                     end
                 else begin
                         Count<=0;
                         DividebyN<=~DividebyN|x;
                      end
             end
         else
            begin
             Clear<=0;
             Count<=Count+1;
            end
     end
endmodule

module controls_the_count(Reset,Clk,Clockref,x,IsnotCorrect,
lower_rate,Clock);
//Set of counter which form the core of the decision making DSP
// These are what decide if the current clock rate is correct or
// if it is wrong
input Clk,Clockref,Reset;
input x;
output IsnotCorrect,lower_rate,Clock;
parameter low_rate_threshold=10'd10;
parameter high_rate_threshold=10'd400;
parameter Clock_bits=6;
parameter freq_bits=3;
parameter Clock_divider_value=7'd19;
parameter
A=3'b000,
B=3'b001,
C=3'b010,
D=3'b011,
E=3'b100;
wire Clr1,clr_state;
reg Inc,Dec,Clr,x1,x2;
wire Clear;
```

```
reg [Clock_bits:0] pd_Inc,pd_Dec,Nreg;
wire [Clock_bits:0] pl_Inc,pl_Dec;
wire Lock;
reg [freq_bits:0] NDec,NInc1,NInc2,NInc;
wire Clear_in,IsnotCorrectt,lower_ratet,Inc_thresh,Dec_thresh;
reg IsnotCorrect,lower_rate;
reg flip;
//The Phase estimator
assign Clr1= (Inc&Dec)|IsnotCorrect;
always @(posedge Clk)
     Clr<=Clr1;
//This is a standard phase-frequency detector
//One FF is controlled by incoming  clock
//and the other FF by the reference clock
//when outputs of both FFs are high we reset the FFs
//Asynchronous Reset
//Asynchronous Clr
always @(posedge Reset or posedge Clock or posedge Clr)
if (Reset) Dec<=0;
else if (Clr) Dec<=0;
else Dec<=1;
always @(posedge Reset or posedge x or posedge Clr)
if (Reset) Inc<=0;
else if (Clr) Inc<=0;
else Inc<=1;
//Phase estimator
always @(posedge Clockref or posedge Clear or posedge Reset)
     if (Reset) pd_Inc<=1;
     else if (Clear) pd_Inc<=1;
     else if (Inc) pd_Inc<=pd_Inc+1;
always @(posedge Clockref or posedge Clear or posedge Reset)
     if (Reset) pd_Dec<=1;
     else if (Clear) pd_Dec<=1;
     else if (Dec) pd_Dec<=pd_Dec+1;
//Frequency estimator
always @(posedge Clear or posedge Reset or posedge Clr)
     if (Reset ) NDec<=0;
     else if (Clr) NDec<=0;
     else if (Dec) NDec<=NDec+1;
     else NDec<=1;
always @(posedge Clockref or posedge Reset or posedge Clr)
     if (Reset ) NInc1<=0;
     else if (Clr) NInc1<=0;
     else if (Inc) NInc1<=NInc1+1;
     else NInc1<=1;
always @(posedge x or posedge Reset or posedge Clr)
     if (Reset ) NInc<=0;
     else if (Clr) NInc<=0;
     else if (Inc) NInc<=NInc+1;
     else NInc<=1;
assign Inc_thresh=(NInc>=3'd5)?1:0;
assign Dec_thresh=(NDec>=4'd10)?1:0;
assign IsnotCorrectt=Inc_thresh|Dec_thresh;
assign lower_ratet = ~NInc[2];
always @(posedge Clk)
begin
     IsnotCorrect<=IsnotCorrectt;
     lower_rate<=lower_ratet;
end
always @(posedge Clockref or posedge Reset)
begin
if (Reset)
     Nreg<=Clock_divider_value;
else if (IsnotCorrect)
     Nreg<=Clock_divider_value;
else //if (x_currstate==E)
     begin
         Nreg<=Clock_divider_value;//x_f-1;
     end
end
check_tran cntrl(Reset,Clockref,NDec,NInc1,Lock,Inc,Dec,Ncntrl);
locker l_1(Reset,Clockref,Clear_in,IsnotCorrect,Ncntrl,pd_Dec,pd_Inc,
pl_Dec,pl_Inc,Lock);
counter_80 c_80(Clockref,Reset,pl_Inc,pl_Dec,Nreg,x,Clock,Clear_in);
clear_delay c_d(Clk,Clear_in,Clear);
endmodule

module check_tran(Reset,Clock,NDec,NInc,Lock,Inc,Dec,Ncntrl);
//This module is responsible for waiting for syncronization sequence
//once lock has been achieved and sending control signals to the rest
//of the system so that the new values can be used to adjust any
//phase errors
input Reset,Lock,Clock,Inc,Dec;
input [3:0] NDec,NInc;
output Ncntrl;
reg Ncntrl;
reg [1:0] sireg,sdreg;
wire [1:0] Ndec,Ninc;
wire NcntrlD,NcntrlI;
wire ignD,ignI,startd,starti,IncDec,Clk;
wire Nd00,Nd01,Nd10,Nd11;
wire Ni00,Ni01,Ni10,Ni11;
//The basic premise of the circuit is that Ndec goes from 0 1 2 0 if
//the data is 010 or 101 and this can be used to find out the phase error
//to resynchronize the clock. Any other sequences eg. 0 1 2 3 0 or 0 1 0
//for Ndec imply that the data has no transitions or more than one
//transition in the cycle
//A similiar logic is used for NInc, which is reset if the pattern is 0 1 0
assign ignD=NDec[3]|NDec[2]|(~Lock);
assign ignI=NInc[3]|NInc[2]|(~Lock);
assign Ndec=\{NDec[1]&(~ignD),NDec[0]&(~ignD)\};
assign Ninc=\{NInc[1]&(~ignI),NInc[0]&(~ignI)\};
assign Nd00=(~Ndec[1])&(~Ndec[0]);
assign Nd01=(~Ndec[1])&(Ndec[0]);
assign Nd10=(Ndec[1])&(~Ndec[0]);
assign Nd11=(Ndec[1])&(Ndec[0]);
assign Ni00=(~Ninc[1])&(~Ninc[0]);
assign Ni01=(~Ninc[1])&(Ninc[0]);
assign Ni10=(Ninc[1])&(~Ninc[0]);
assign Ni11=(Ninc[1])&(Ninc[0]);
assign startd=Nd01&(~Inc);
assign starti=Ni01&(~Dec);
always @(posedge Clock)
if (Reset)
     sdreg=2'b00;
else
case (sdreg)
     2'b00:if (startd) sdreg=2'b01;
     2'b01:if (Nd10) sdreg=2'b10;
            else if (Nd00) sdreg=2'b11;
     2'b10:if (Nd00) sdreg=2'b11;
            else if (Nd11) sdreg=2'b00;
          else sdreg=2'b10;
     2'b11:if (Nd01) sdreg=2'b00;
     default: sdreg=2'b00;
endcase
always @(posedge Clock)
if (Reset) sireg=2'b00;
else
case (sireg)
     2'b00:if (starti) sireg=2'b01;
     2'b01:if (Ni10) sireg=2'b10;
```

```
           else if (Ni00) sireg=2'b00;
     2'b10:if (Ni00) sireg=2'b11;
//           else if (Ni11) sireg=2'b00;
         else sireg=2'b10;
     2'b11:if (Ni01) sireg=2'b00;
     default: sireg=2'b00;
endcase
assign NcntrlD=(sdreg[1]|sdreg[0])&(~ignD);
assign NcntrlI=(sireg[1]&sireg[0])&(~ignI);
always @(Nd00 or Ni00)
if (Nd00 & Ni00)  Ncntrl<=(NcntrlD)|(NcntrlI);
else if (Nd00 & (~Ni00)) Ncntrl<=(NcntrlD);
else if (Ni00 & (~Nd00)) Ncntrl<=(NcntrlI);
else Ncntrl<=0;
endmodule

module clear_delay(Clk,Clear_in,Clear);
//A 1 unit delay for the clear signal
input Clk,Clear_in;
output Clear;
reg Clear,Clear1;
always @(posedge Clk)
begin
     Clear<=Clear_in;
     Clear1<=Clear;
end
endmodule

module locker(Reset,Clk,Clear_in,Changef,Ncnrtl_cnt,NDec,NInc,NDout,NIout
,Lock);
//This module takes care of the locking criteria
//It does so by assuming a shift register to store patterns of phase errors
//and setting lock depending on the duration for which the phase error
// is below a threshold of 3
input Clear_in,Reset,Clk,Changef,Ncnrtl_cnt;
input [6:0] NDec,NInc;
output [6:0] NDout,NIout;
output Lock;
reg [2:0] countI,countD;
reg [6:0] Nd1,Nd2,Nd3,Nd4;
reg [6:0] Ni1,Ni2,Ni3,Ni4;
wire countI_cnt,countD_cnt,LockI,LockD,Lock,Rst_Clr,Clr_Chg,Cnt1,Cnt0;
wire [6:0] NDout,NIout;
reg Clear;
```

```
assign Rst_Clr=Changef|Reset;
assign Clr_Chg=Changef|Clear_in;
assign Cnt1=(~Lock)|(Ncnrtl_cnt);
assign Cnt0=(~Clear);
//These multiplexor feed in the values of phase errors to the counter_80
// module depending on the state of the system
//If the signal has been locked the value is the one stored in the shift
//register until there is a syncronization sequence sent in by check_tran
//module. If the lock has not been achieved it sends the actual phase error
//computed by the controls_the_count module
multiplexor4_1 md0(NDout[0],Nd1[0],1'b1,NDec[0],NDec[0],Cnt1,Cnt0);
multiplexor4_1 md1(NDout[1],Nd1[1],1'b0,NDec[1],NDec[1],Cnt1,Cnt0);
multiplexor4_1 md2(NDout[2],Nd1[2],1'b0,NDec[2],NDec[2],Cnt1,Cnt0);
multiplexor4_1 md3(NDout[3],Nd1[3],1'b0,NDec[3],NDec[3],Cnt1,Cnt0);
multiplexor4_1 md4(NDout[4],Nd1[4],1'b0,NDec[4],NDec[4],Cnt1,Cnt0);
multiplexor4_1 md5(NDout[5],Nd1[5],1'b0,NDec[5],NDec[5],Cnt1,Cnt0);
multiplexor4_1 md6(NDout[6],Nd1[6],1'b0,NDec[6],NDec[6],Cnt1,Cnt0);
multiplexor4_1 m0(NIout[0],Ni1[0],1'b1,NInc[0],NInc[0],Cnt1,Cnt0);
multiplexor4_1 m1(NIout[1],Ni1[1],1'b0,NInc[1],NInc[1],Cnt1,Cnt0);
multiplexor4_1 m2(NIout[2],Ni1[2],1'b0,NInc[2],NInc[2],Cnt1,Cnt0);
multiplexor4_1 m3(NIout[3],Ni1[3],1'b0,NInc[3],NInc[3],Cnt1,Cnt0);
multiplexor4_1 m4(NIout[4],Ni1[4],1'b0,NInc[4],NInc[4],Cnt1,Cnt0);
multiplexor4_1 m5(NIout[5],Ni1[5],1'b0,NInc[5],NInc[5],Cnt1,Cnt0);
multiplexor4_1 m6(NIout[6],Ni1[6],1'b0,NInc[6],NInc[6],Cnt1,Cnt0);
always @(posedge Clr_Chg or posedge Reset)
if (Reset) Clear<=1;
else Clear<=~Clear;
always @(posedge Clear or posedge Reset)
begin
     if (Reset)
         begin
               Nd1<=0;
             Nd2<=0;
             Nd3<=0;
             Nd4<=0;
         end
     else
         begin
             if (Lock) Nd1<=Nd4;
             else Nd1<=NDec;
             Nd2<=Nd1;
             Nd3<=Nd2;
             Nd4<=Nd3;
         end
```

```
end
always @(posedge Clear or posedge countD_cnt)
begin
     if (countD_cnt) countD<=2'd0;
     else if (countI_cnt) countD<=2'd0;
     else if (LockD) countD <=3'd7;
     else countD<=countD+1;
end
assign countD_cnt=(Nd1[6]|Nd1[5])|(Nd1[4]|Nd1[3])|Rst_Clr;
assign LockD=countD[2];
always @(posedge Clear or posedge Reset)
begin
     if (Reset)
         begin
               Ni1<=0;
             Ni2<=0;
             Ni3<=0;
             Ni4<=0;
         end
     else
         begin
             if (Lock) Ni1<=Ni4;
             else Ni1<=NInc;
             Ni2<=Ni1;
             Ni3<=Ni2;
             Ni4<=Ni3;
         end
end
always @(posedge Clear or posedge countI_cnt)
begin
     if (countI_cnt) countI<=2'd0;
     else if (countD_cnt) countI<=2'd0;
     else if (LockI) countI <=3'd7;
     else countI<=countI+1;
end
assign countI_cnt=(Ni1[6]|Ni1[5])|(Ni1[4]|Ni1[3])|Rst_Clr;
assign LockI=countI[2];
assign Lock=LockI&LockD;
endmodule

module multiplexor4_1(out, in1, in2, in3, in4, cntrl1, cntrl2);
output out;
input in1, in2, in3, in4, cntrl1, cntrl2;
wire notcntlr1, notcntrl2, w, x, y, z;
```

```
not (notcntrl1, cntrl1);
not (notcntrl2, cntrl2);
and (w, in1, notcntrl1, notcntrl2);
and (x, in2, notcntrl1, cntrl2);
and (y, in3, cntrl1, notcntrl2);
and (z, in4, cntrl1, cntrl2);
or (out, w, x, y, z);
endmodule

module sets_the_bypass(Clock,Reset,x,Bypass);
// this module is the controller for the bypass unit
// This waits for the first data sample to arrive before starting the
// counters which actually do the computation, is useful to reduce power
input Reset,Clock;
input x;
output Bypass;
reg  current_state;
wire  next_state;
//FSM
//State Table
//Reset     S(n)     x     S(n+1)     Bypass
//0     0     0     0     0
//0     0     1     1     1
//0     1     x     1     1
//1     x     x     0     0
always @(posedge Clock or posedge Reset)
     if (Reset)
         current_state=0;
     else
         current_state=next_state;
assign Bypass=current_state;
assign next_state=\{(x|current_state)\};
endmodule

module simple(clock,reset,x,y);
//this module is a simple FSM which acts like an input filter
//It removes all single bit sequences from the input
input x,clock,reset;
output y;
parameter
A=2'b00,
B=2'b01,
C=2'b10,
D=2'b11;
```

```
parameter N=2'd3;
reg[1:0] currstate,nextstate;
reg[1:0] Count;
wire test;
reg y,ybar;
//2 BIT Counter used by the FSM
//Asynchronous Reset
always @(posedge clock or posedge ybar)
if (ybar)
     Count<=2'd0;
else
     Count<=Count+1;
//Control register for the Counter
//Since y is a combinatorial logic output ,subject to change
//we need to make sure that the reset is not an Asyncronous event
always @(posedge clock)
     if (reset)
         ybar<=1;
     else ybar<=~y;
// This is the basic FSM for simple module
// It waits for 2 1's to occur in consecutive cycles and once that occurs
// it sets an output to high
// The output remains high for 4 clock cycles before going low
always @(posedge clock) // Basic FSM state changer
     if (reset)
           currstate<=A;
     else
         currstate<=nextstate;
//State Machine
// State Table is as follows
// q(n) (x) (Count==3) q(n+1)  Y     COUNT
// A     0     x     A     0     0
// A     1     x     B     0     0
// B     0     x     A     0     0
// B     1     X     C     0     0
// C     X     X     D     1     COUNT++
// D     X     0     D     1     COUNT++
// D     X     1     A     0     0
always @(currstate or x or test)
     case (currstate)
     A: if (x)
         begin
         nextstate=B;y=0;
         end
         else
              begin
         y=0;
         nextstate=A;
         end
      B:if (x)
         begin
         nextstate=C;
         y=0;
         end
         else begin
         y=0;nextstate=A;
         end
     C:begin nextstate=D;y=1; end
     D:if (test)
         begin
         nextstate=A;
         y=0;
         end
       else begin
            nextstate=D;
         y=1;
         end
     default: begin
         nextstate=A;
         y=0;
         end
     endcase
assign test=&Count;  //control logic to check for Count=2'b11;
endmodule
}
```

A.2 Synposys Script

```
read -f verilog clockrecoverer.v
read -f verilog controls_the_count.v
read -f verilog counter_10.v
read -f verilog counter_4.v
read -f verilog counter_80.v
read -f verilog multiplexor4_1.v
read -f verilog sets_the_bypass.v
read -f verilog simple.v
read -f verilog clockbyN.v
read -f verilog check_tran.v
read -f verilog locker.v
read -f verilog clear_delay.v
target_library = {"ms080cmosxCells_XXW.db"}
link_library = {"ms080cmosxCells_XXW.db"}
set_operating_conditions -library "ms080cmosxCells_XXW" "T125_V4.5"
//Set dont use list
//The list of cells in 0.35um SOI is limited and hence we need to make
//sure that cells which dont exist are not used.
set_dont_use ms080cmosxCells_XXW/PULLUP
set_dont_use ms080cmosxCells_XXW/PULLDWN
set_dont_use ms080cmosxCells_XXW/AND2X2
set_dont_use ms080cmosxCells_XXW/AND2X3
set_dont_use ms080cmosxCells_XXW/AND2X4
set_dont_use ms080cmosxCells_XXW/BUFX2
set_dont_use ms080cmosxCells_XXW/INVX16
set_dont_use ms080cmosxCells_XXW/INVX2
set_dont_use ms080cmosxCells_XXW/INVX4
set_dont_use ms080cmosxCells_XXW/INVX8
set_dont_use ms080cmosxCells_XXW/NAND2X2
set_dont_use ms080cmosxCells_XXW/NAND2X3
set_dont_use ms080cmosxCells_XXW/NAND2X3
set_dont_use ms080cmosxCells_XXW/NOR2X2
set_dont_use ms080cmosxCells_XXW/NOR2X3
set_dont_use ms080cmosxCells_XXW/NOR2X4
set_dont_use ms080cmosxCells_XXW/OR2X2
set_dont_use ms080cmosxCells_XXW/OR2X3
set_dont_use ms080cmosxCells_XXW/XOR2X2
set_dont_use ms080cmosxCells_XXW/AND3X2
set_dont_use ms080cmosxCells_XXW/AND4X2
set_dont_use ms080cmosxCells_XXW/BUFX4
set_dont_use ms080cmosxCells_XXW/INVX12
set_dont_use ms080cmosxCells_XXW/NAND3X2
set_dont_use ms080cmosxCells_XXW/NAND4X2
set_dont_use ms080cmosxCells_XXW/OR2X4
set_dont_use ms080cmosxCells_XXW/OR3X2
set_dont_use ms080cmosxCells_XXW/OR4X2
set_dont_use ms080cmosxCells_XXW/BUFX12
set_dont_use ms080cmosxCells_XXW/BUFX16
set_dont_use ms080cmosxCells_XXW/BUFX20
set_dont_use ms080cmosxCells_XXW/BUFX8
set_dont_use ms080cmosxCells_XXW/INVX20
set_dont_use ms080cmosxCells_XXW/DEC4
set_dont_use ms080cmosxCells_XXW/DEL1
set_dont_use ms080cmosxCells_XXW/DEL2
set_dont_use ms080cmosxCells_XXW/DFFR_SCN
set_dont_use ms080cmosxCells_XXW/DFFSR
set_dont_use ms080cmosxCells_XXW/DSEL4
set_dont_use ms080cmosxCells_XXW/EDFF
set_dont_use ms080cmosxCells_XXW/MAJ3N
set_dont_use ms080cmosxCells_XXW/NDLAT
set_dont_use ms080cmosxCells_XXW/OAI21
set_dont_use ms080cmosxCells_XXW/OR3
set_dont_use ms080cmosxCells_XXW/OR4
set_dont_use ms080cmosxCells_XXW/TBUF
set_dont_use ms080cmosxCells_XXW/TBUFX2
set_dont_use ms080cmosxCells_XXW/TBUFX4
set_dont_use ms080cmosxCells_XXW/TINV
set_dont_use ms080cmosxCells_XXW/TLATR
set_dont_use ms080cmosxCells_XXW/AND5
set_dont_use ms080cmosxCells_XXW/AND6
set_dont_use ms080cmosxCells_XXW/AOI211
set_dont_use ms080cmosxCells_XXW/AOI221
set_dont_use ms080cmosxCells_XXW/AOI222
set_dont_use ms080cmosxCells_XXW/AOI311
set_dont_use ms080cmosxCells_XXW/AOI321
set_dont_use ms080cmosxCells_XXW/AOI322
set_dont_use ms080cmosxCells_XXW/DFFSR_SCN
set_dont_use ms080cmosxCells_XXW/DFFS_SCN
set_dont_use ms080cmosxCells_XXW/DFF_SCN
set_dont_use ms080cmosxCells_XXW/EDFFS
set_dont_use ms080cmosxCells_XXW/EDFFS_SCN
set_dont_use ms080cmosxCells_XXW/EDFF_SCN
set_dont_use ms080cmosxCells_XXW/FADD
set_dont_use ms080cmosxCells_XXW/GPULD
set_dont_use ms080cmosxCells_XXW/GPULU
set_dont_use ms080cmosxCells_XXW/HADD
set_dont_use ms080cmosxCells_XXW/JKFFS
set_dont_use ms080cmosxCells_XXW/JKFF_SCN
set_dont_use ms080cmosxCells_XXW/NAND5
set_dont_use ms080cmosxCells_XXW/NAND6
set_dont_use ms080cmosxCells_XXW/NOR4
set_dont_use ms080cmosxCells_XXW/NOR4X2
set_dont_use ms080cmosxCells_XXW/NOR5
set_dont_use ms080cmosxCells_XXW/NOR6
set_dont_use ms080cmosxCells_XXW/NRLAT
set_dont_use ms080cmosxCells_XXW/OAI211
set_dont_use ms080cmosxCells_XXW/OAI22
set_dont_use ms080cmosxCells_XXW/OAI221
set_dont_use ms080cmosxCells_XXW/OAI31
set_dont_use ms080cmosxCells_XXW/PWR_RESET
set_dont_use ms080cmosxCells_XXW/TBUFX12
set_dont_use ms080cmosxCells_XXW/TBUFX8
set_dont_use ms080cmosxCells_XXW/TINVX2
set_dont_use ms080cmosxCells_XXW/TLAT
set_dont_use ms080cmosxCells_XXW/TLATS
set_dont_use ms080cmosxCells_XXW/XNOR2X2
set_dont_use ms080cmosxCells_XXW/XOR3
set_dont_use ms080cmosxCells_XXW/XOR3X2
set_dont_use ms080cmosxCells_XXW/AND7
set_dont_use ms080cmosxCells_XXW/AND8
set_dont_use ms080cmosxCells_XXW/AOI21
set_dont_use ms080cmosxCells_XXW/AOI22
set_dont_use ms080cmosxCells_XXW/AOI31
set_dont_use ms080cmosxCells_XXW/AOI32
set_dont_use ms080cmosxCells_XXW/AOI33
set_dont_use ms080cmosxCells_XXW/AOI331
set_dont_use ms080cmosxCells_XXW/AOI332
set_dont_use ms080cmosxCells_XXW/AOI333
set_dont_use ms080cmosxCells_XXW/BUS_COND
set_dont_use ms080cmosxCells_XXW/EDFFR
set_dont_use ms080cmosxCells_XXW/EDFFR_SCN
set_dont_use ms080cmosxCells_XXW/EDFFSR
set_dont_use ms080cmosxCells_XXW/EDFFSR_SCN
set_dont_use ms080cmosxCells_XXW/JKFFR_SCN
set_dont_use ms080cmosxCells_XXW/JKFFSR
set_dont_use ms080cmosxCells_XXW/JKFFSR_SCN
set_dont_use ms080cmosxCells_XXW/JKFFS_SCN
set_dont_use ms080cmosxCells_XXW/NAND7
set_dont_use ms080cmosxCells_XXW/NAND8
set_dont_use ms080cmosxCells_XXW/NOR7
set_dont_use ms080cmosxCells_XXW/NOR8
set_dont_use ms080cmosxCells_XXW/OAI222
set_dont_use ms080cmosxCells_XXW/OAI311
set_dont_use ms080cmosxCells_XXW/OAI32
set_dont_use ms080cmosxCells_XXW/OAI321
set_dont_use ms080cmosxCells_XXW/OAI322
set_dont_use ms080cmosxCells_XXW/OAI33
set_dont_use ms080cmosxCells_XXW/OAI331
set_dont_use ms080cmosxCells_XXW/OAI332
set_dont_use ms080cmosxCells_XXW/OAI333
set_dont_use ms080cmosxCells_XXW/OR5
set_dont_use ms080cmosxCells_XXW/OR6
set_dont_use ms080cmosxCells_XXW/OR7
set_dont_use ms080cmosxCells_XXW/OR8
set_dont_use ms080cmosxCells_XXW/TBUFX16
set_dont_use ms080cmosxCells_XXW/TBUFX20
set_dont_use ms080cmosxCells_XXW/TINVX12
set_dont_use ms080cmosxCells_XXW/TINVX16
set_dont_use ms080cmosxCells_XXW/TINVX20
set_dont_use ms080cmosxCells_XXW/TINVX4
set_dont_use ms080cmosxCells_XXW/TINVX8
set_dont_use ms080cmosxCells_XXW/TLATSR
set_dont_use ms080cmosxCells_XXW/XNOR3
set_dont_use ms080cmosxCells_XXW/XNOR3X2
current_design = clockrecoverer
uniquify
set_max_area 0
compile -incremental
report_timing
```

A.3 MATLAB Scripts

```
%DDBPSK demodulator
function y= DDBPSKd(x,k);
y=[];
th=pi/2-(pi/2)*sign(x);
for i=1:(length(x)-(k+1))
    thd=(th(i+k+1)-th(i+k))-(th(i+1)-th(i));
   ybar=(thd-pi/2)*2/pi;
   y=[y -cos(thd)];
end

%DDBPSK modulator
function [A,y]=DDBPSKm(x,SNR,k);
A=sqrt(2)*((10)^(SNR/20));
thd=x*pi/2 + pi/2;
y=[pi*ones(1,k+1)];
for i=1:length(x)
   th=abs(thd(i)+y(i+k)+y(i+1)-y(i));
   y=[y th];
end

%Doppler channel model
function [sys,x0,str,ts] = sfundsc1(t,x,u,flag,Ts,dmax,ddmax)
```

```
switch flag,
  % Initialization %
  case 0,
   [sys,x0,str,ts]=mdlInitializeSizes(Ts,dmax,ddmax);
  % Update %
  case 2,
    sys = mdlUpdate(t,x,u,Ts,dmax,ddmax);
  % Output %
  case 3,
    sys = mdlOutputs(t,x,u,Ts,dmax,ddmax);
  % Terminate %
  case 9,
    sys = [];
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end
%end sfundsc1
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the
% S-function.
function [sys,x0,str,ts]=mdlInitializeSizes(Ts,dmax,ddmax)
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 2;
sizes.NumOutputs     = 2;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 2;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [0 2*(rand-1)*dmax];%4*pi*(rand-0.5)*dmax*Ts;
str = [];
ts  = [Ts 0]; % Ts set by user
% end mdlInitializeSizes
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
function sys = mdlUpdate(t,x,u,Ts,dmax,ddmax)
ph=x(1);
dop=x(2);
ph=ph+2*pi*dop*Ts; % new phase = old phase + increment
dop=dop+ddmax; % new doppler  = old doppler  + change;
if (dop>dmax) dop=dmax;
elseif (dop<-dmax) dop=-dmax;
end
```

```
sys=[ph;dop];
%end mdlUpdate
% mdlOutputs
% Return the output vector for the S-function
function sys = mdlOutputs(t,x,u,Ts,dmax,ddmax)
ph=x(1);
if (u~=[0 0]')
   th=rand*2*pi;
   sys=0.005*[cos(th) sin(th)]';
end
sys = [u(1)*cos(ph)-u(2)*sin(ph) u(1)*sin(ph)+u(2)*cos(ph)]';
%end mdlOutputs


%timing simulation data generator for Simulink
clear all;clc
upfreq=4;
N=32;
Fk=100000;
Fc=2e6;  %carrier freq
Fs=upfreq*Fc;%sampling rate
Tk=1/Fk;
Tc=1/Fc;
Tb=1/Fk;
Ts=1/Fs;
dmax=10000;
ddmax=10;
upfreq=4; %upsample rate
len=10;
SNR=120;
ain=ones(1,len);
b=[ones(1,10) -ones(1,10)];
b=[b b b b b];
c=abs(b);
ain(1:2:len)=-1;
ain=[ ain ain ain ain ain ain ain ain ain ain];
ain=[-ones(1,10) ain ain ain ain ain ain ain ain]
len=length(ain);
grid off
sim('timingoldver');
yout=yout(:);
yout=sign(yout);
save -ascii data.mxt yout
```

```
%reciever simulator for Simulink
clc;
clear all
upfreq=4; %upsample rate
len=10;  % no of bits of data
x=randsrc(len,1,[],1235);
dmax=10000;
ddmax=10;
Fk=100e3;
Fc=2e6;  %carrier freq
Fb=100e3; %bitrate
Fs=upfreq*Fc;%sampling rate
N=Fs/Fb;% downsample rate
Tb=1/Fb;
Ts=1/(Fs);
Tc=1/(Fc);
Tk=1/Fk;
[len Fb Fc Fs]
snr=[14 10 5];
BER=[0 0 0];
[A,simin]=DDBPSKm(x,1,1);
simin=[cos(simin) -1];
format long
for i=1:length(snr)
SNR=snr(i);
sim('reciverold');
temp(1:len)=-yout(4:3+len);
BER(i)=length(find(temp~=x'))/len;
end


%performance comparison between DDPSK1, DDPSK2 and FSK
clear all;
%loop for SNR
format long
bersnr1=[];
bersnr2=[];
Runs = 10000;
k=2;
for SNR = 2:0.1:6
   b=rand(1,Runs);
   noise1 = randn(1,Runs+k);
   noise2 = randn(1,Runs+k+1);
   signal=sign(b-0.5);
   k=1;
   transmitted1=DDBPSKm(signal,SNR,k);
   recieved1=transmitted1+noise1;
   predicted1=DDBPSKd(recieved1,k);
   BER1 = length(find(predicted1~=signal))/length(signal);
   bersnr1=[bersnr1 BER1];
   k=2;
   transmitted2=DDBPSKm(signal,SNR,k);
   recieved2=transmitted2+noise2;
   predicted2=DDBPSKd(recieved2,k);
   BER2 = length(find(predicted2~=signal))/length(signal);
   bersnr2=[bersnr2 BER2];
end
SNR=2:0.1:6;
bertheo=0.5*erfc(sqrt(SNR));
save bersnr1
save bersnr2
save bertheo
semilogy(SNR,bersnr1,'b',SNR,bersnr2,'r',SNR,bertheo,'g-.');
```

Bibliography

[1] L.M. Davis, I.B. Collings, R.J. Evans, “Estimation of LEO satellite channels ”*Proceedings of 1997 International Conference on Information, Communications and Signal Processing, 1997. ICICS*” Vol. 1, pp. 15-19, 9-12 Sept. 1997.

[2] Aura Ganz, Yebin Gong, Bo Li, “Performance study of low earth orbit satellite systems ”,*IEEE transaction on communication* Vol. 42, pp. 1866-1871, Feb 1994.

[3] Cheng-Ying Yang, Junghwan Kim, ”Performance analysis of low earth orbit (LEO) land mobile satellite using moment technique”, *IEEE Proceeding Military Communications Conference, 1998, MILCOM 98* Vol. 3, pp. 883-887, 18-21 Oct. 1998.

[4] I. Ali, N. Al-Dhahir, J.E. Hershey, Doppler characterization for LEO satellites,*IEEE Transactions on Communications* Vol. 46, Issue 3, pp 309-313, March 1998.

[5] E. Vilar, J. Austin, “Analysis and correction techniques of Doppler shift for nongeosynchronous communication satellites”, *International Journal of Satellite Communication*, Vol. 9, pp. 123-126, 1991.

[6] M. Changning, W. Dongjin, The performance of DDPSK over LEO mobile satellite channels.

[7] M. K. Simon and M. H. Sami, Digital Communication Techniques: Signal Design and Detection. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[8] FSK : Signals and Demodulation, http://rfwireless.rell.com/pdfs/TN_WJfsk.pdf

[9] Non-Coherent Detection, www.ntu.edu.sg/home/asysgao/SC205-6-P2.pdf

[10] Rafferty W. and Divsalar D., " Modulation and coding for land mobile satellite channels, " *IEEE Proc. ICC88* June 1988, pp.1105-1111.

[11] M.K. Simon, D. Divsalar, "Doppler-corrected differential detection, "*IEEE Transactions on Communications* Vol. 37, pp 99-109, Feb 1989.

[12] D. Divsalar, M.K. Simon, "Multiple-Symbol differential detection of MPSK, "*IEEE transactions on communication* Vol. 38, pp. 300-308, Mar 1990.

[13] M.K.Simon and D. Divasalar, " The implementation and performance of single and double differential detection schemes, "*IEEE transactions on communication* , pp. 278-291, Feb 1992.

[14] Fulvio Gini and Georgios B. Giannakis, " Generalized Differential Encoding: A Nonlinear Signal Processing Perspective, "*IEEE Transactions on Signal Processing* Vol. 46, No. 11, Nov. 1998.

[15] D.K. Alphen, W.C. Lindsey, "Higher-order differential phase shift keying, "*IEEE Transactions on comunication* Vol. 42, pp. 440-448, Feb. 1994.

[16] N.Hamamoto, "Differential detection with IIR filter for improving DPSK detection performance, "*IEEE Trnasactions on Communications* Vol. 44, pp. 959-966, August 1996.

[17] J.L Buetefuer and W.G Cowley, " A frequency insensitive demodulator for double differential PSK, "*DASP 2001*

[18] Simon M.K. and Lindsdey W.C., Digital Communication Technique. Englewood Cliffs, NJ: Prentice-Hall,1995.

[19] Grayver, E., Daneshrad, B., " A low-power all-digital FSK receiver for space applications, "*IEEE Transactions on Communications* Vol. 49, Issue 5, May 2001. pp 911-921

[20] Wu, P.H., " The optimal power BPSK demodulator with a 1-bit front-end, "*Proc. MILCOM'98* Vol. 3.3, 1998. pp 730-735

[21] W. C. Lindsey and C. M. Chie, " A Survey of digital phase-locked loops, "*Proc. IEEE* 69, 4, pp.410-431, April 1981

[22] Using Field programmable gate arrays for Digital PLL applications, www.actel.com/appnotes/s04_18.pdf

[23] Rhode, U.L., Digital PLL Frequency Synthesizers, Theory and Design. Englewood Cliffs, NJ: Prentice Hall, 1983

[24] Gardner, F.M., Phase lock Techniques, 2nd Ed. New York: Wiley, 1979

[25] Lindsey, W.C., Phase-Locked Loops and their applications. New York: IEEE Press

[26] Huang, T.H.D., Zencir E., Yuce M.R., Dogan N.S.,Liu W. and Arvas E., "A 22-mW 435MHz silicon on insulator CMOS high-gain LNA for subsampling receivers," *IEEE ISCAS03 in press*

[27] Zencir E., Dogan N.S. and Arvas E.," A low-power UHF RF frontend for a low-IF receiver," *IEEE ASIC/SOC02* , pp.331-335.

[28] Yuce M.R., Liu W., Damiano J., Bharath B., Franzon P.F., Dogan N.S., " A Low-Power PSK receiver for space applications in 0.35um SOI CMOS"

[29] Simulink Model-Based and System-Based Design - writing S functions

[30] Colinge J.P., Silicon-on-Insulator Technology: Materials to VLSI. Kluwer Academic Publishers,1997.