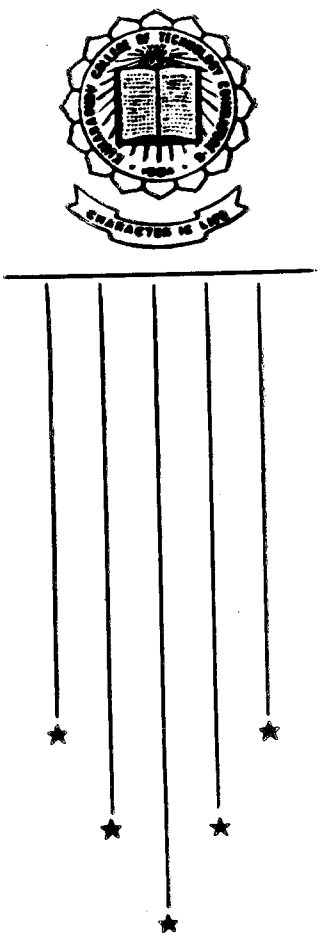
A FSK Demodulator, Based on DPLL Techniques Implemented Using DSP

PROJECT REPORT



Submitted by

S. MOHAN

K.R. SEVUGAN

A. SREEDHAR

J. BARAKATH ALI

Guided by

Mrs. H. MANGALAM, M.E.,

Sponsored by

Vikram Sarabhai Space Centre, ISRO
Trivandrum

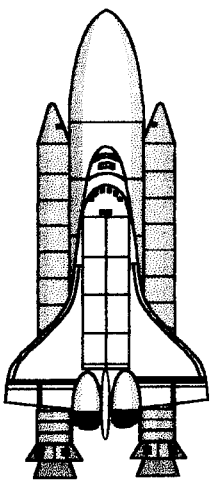
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF
BACHELOR OF ENGINEERING IN
ELECTRONICS & COMMUNICATION ENGINEERING
OF THE BHARATHIAR UNIVERSITY

1997 - 98

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Kumaraguru College of Technology

COIMBATORE - 641 006.



Certificate


50

ACKNOWLEDGEMENT

We thank our Principal **Dr.S.Subramanian, B.E., M.Sc.(Engg), Ph.D., S.M., IEEE**, for his kind patronage and encouragement.

Our thanks to our Head of the Department, **Prof.M.Ramasamy, M.E., MIEEE (USA), M.I.E, MISTE** for his support throughout the project.

We express our sincere thanks to our beloved guide **Mrs.H.Mangalam, M.E., MISTE,** Senior Lecturer, Department of Electronics and Communication Engineering for her guidance and encouragement. We are very grateful to her, for her technical ideas which made this project success.

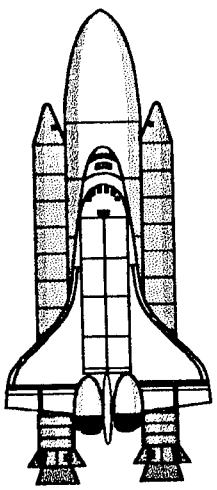
We extend our sincere gratitude to **Mr.T.V.Narayana,** Head BTTCS, BSD, ISRO, Trivandrum for his valuable guidance during the course of our project.

This project was carried out under the overall guidance of **Mr.P.Vinod,** BTTCS, BSD, ISRO. We also acknowledge his help in the preparation of this project.

We extend our thanks to **Mr.L.Muthu,** Head, GSLV Project, VSSC, ISRO, Trivandrum who get this project for us.

We profusely thank **Mr.Jothi Ramalingam, Mr.Anuroop,** and **Mrs.Fineetha,** Engineers of VSSC, ISRO, who rendered it possible for us to complete this project. They had helped us unflaggingly at any stages of the project despite a tight schedule.

We extend our thanks to all the staff members and students friends who are helpful in completing the project.



Synopsis

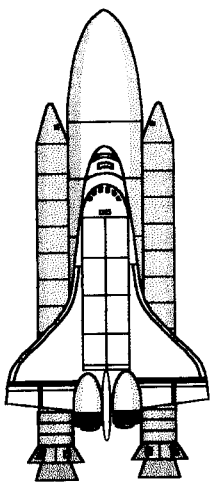
SYNOPSIS

This project deals about the demodulation of a frequency shift keyed signal using a high-speed microcomputer ADSP 2101. Digital phase locked loop is implemented in ADSP 2101. The concept of FSK demodulation is simulated in computer using C language and realized in real time using digital signal processing chip 2101.

The digital phase locked loop deserves particular interest because it can be used in time sharing operation and its characteristics can easily be controlled.

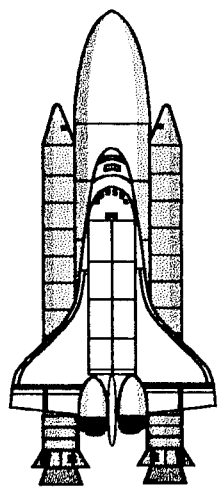
The frequency shift keyed signals are fed to the analog to Digital converter. The analog signals are converted into digital format. The digitized sign is filtered using loop filter and then, they are added with PLL's free running frequency samples. These added samples are given to the phase detector, according to the phase error variation the outputs are obtained. This is done, so that error is less and the signal is decoded properly.

This FSK demodulator is used in reception side of tele-command system.



Contents

CONTENTS



Introduction

1. INTRODUCTION

This project forms a portion of the tele-command decoder which is a subsystem used in launch vehicles. The data which is sent from the ground station should be received without any error. Therefore the data are encoded with convalutional codes, one of the efficient coding methods. After coding, the data are FSK modulated.

The aim of the project is to develop a FSK demodulation using DSP technique, to be used in reception side of the tele command system. The concept and idea of FSK demodulation is first simulated in computer using C language, then FSK demodulation is realized in real time using DSP chip ADSP 2101.

ADSP 2101 is a single chip microcomputer optimized for digital signal processing and other high-speed numeric processing applications. It contains three computational units, data address generators and a program sequencer with two serial parts, a programmable timer, interrupt capabilities and a data memory RAM.

Phase Locked Loops (PLL) all used widely in electronic circuits, communication equipment etc., With the development of modern digital technology, phase locked loops are also becoming digitized. The digital signal processing type digital PLL (DPLL) realized by microprocessors and software deserves it can be used in time-sharing operation and its characteristics can easily be controlled. However, the DSP type DPLL has been derived from an analog PLL by directly realizing its operation with DSP methods.

Digital signal processing involves the representation, transmission and manipulation of signals using numerical techniques and digital processors. It has been an exciting and growing technology during the past few years. Its applications have also been expanded vigorously to encompass various areas of signal processing.

In designing a DSP application, the following immediate technical challenges are involved.

- Selecting digital signal processors powerful enough to perform the task.
- Obtaining technical supports as well as development tools.
- Creating DSP algorithms to execute the application.
- Implementing these algorithms to execute the application.

While more DSP algorithms are being discovered, better tools are also being developed to implement these algorithms. One of the most important break through in electronic technology is the high speed digital signal processors. These are essentially high-speed microprocessors designed specifically to perform computation intensive DSP algorithms. By taking the advantage of advanced architecture, parallel processing and dedicated DSP instruction sets combined with floating point operations, these devices can execute millions of DSP operation per second.

Digital signal processors provide a better solution than their analog counterparts for reasons of reliability, reproducibility, compactness and efficiency.

Digital signal processing is used for demodulation of FSK signals, due to its described advantages:

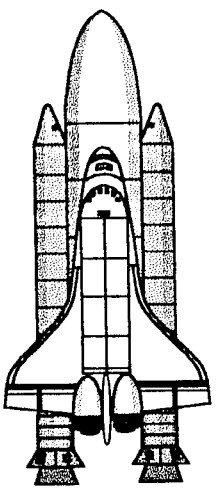
- High speed of operation.
- Software implementation.
- High speed algorithms that can be used only with DSP.

A typical digital signal processor starts with digitization (sampling) of a signal at a rate that is atleast twice the highest significant frequency of the signal. The remaining part of analog-digital conversion is quantization of the discrete data, which, unlike sampling, is an irreversible process. The resulting datas are processed digitally by recursive on non-recursive filtering, transformations, and the combination of various techniques so that the deserved information can be extracted on enhanced.

The quality of data collected by a data acquisition system depends on the signal to Noise ration and on resolution of Analog to Digital Converter is used. Contributions to noise can include the following noises.

Amplifies Noise, Alaised source noise, Broadband source noise (which passes through the noise bandwidth of signal conditioner) and residual common mode noise at the output of instrumentation amplifies.

If minimum input signal to noise ratio is not achieved, post processing of digital data could be used to improve signal to noise ratio.



Digital Communication Systems

2. DIGITAL COMMUNICATION SYSTEMS

2.1 INTRODUCTION:

Digital communication systems are now in common use and give every indication if it eventually sweeping away even such time honoured analog communication systems as are now used in radio, telephone and television.

2.2 DIGITAL COMMUNICATION DISCRIPTION:

In Digital communication, the continuous signal should be converted into discrete information, by means if sampling and quantisation. The discrete information source as finite O value. The voltage levels of symbols are defined in the discrete information source. The sampling of the analog signal is shown in the figure 2.1(a), 2.1(b) and 2.1(c).

2.3 SAMPLING AND NYQUIST CRITERIA:

The sampling should be done according to Nyquist criteria. The sampling rate f_s >2 fm. Where fm is the maximum frequency of the band limited signal. If this is not satisfied the overlapping and the aliasing will occur.

Digital communication is unique in that a communication channel is specified by the number of bits per second it can carry. The nature of information is unimportant to the communication systems. This is in marked contrast to existing analog systems, in which the properties of the transmitted waveform are inextricably tied up with the properties of the application, and very weakly tied to considerations of communication channel. This means that the digital communication system is one system that is designed to best use a given channel and an analog system is one designed to best use a given sources. The spectrum of a well designed digital communication waveform is a good match to the pass band characteristics of the channel.

2.4 PROBLEMS IN DIGITAL COMMUNICATION:

The ever increasing demand for digital transmission channels in the Radio Frequency band has resulted in spectral congestion that is likely to cause adjacent and co-channel interference problems. Some solutions to this problem include: -

a) New allocations at high frequencies.

b) Better management of existing allocation.

c) Use of frequency re-use techniques.

d) Use of spectrally efficient modulation schemes.

A general communication system design objective is to use the primary communication resources viz., transmitted power and channel bandwidth as efficiency as possible. Depending on which of these is more previous channels can be classified as power limited or band limited spectrally efficient modulation techniques are used in band limited channels to save band width.

2.5 ADVANTAGES OF DIGITAL COMMUNICATION:

The performance indicator for an analog communication system is usually the output SNR while for a digital communication system, the performance is generally measured in terms of probability of bit error (Pe) or bit error rate. The advantages in digital communication are

1. Error manipulation is possible.
2. Signal manipulation is possible.
3. Greater dynamic range.
4. Security of Transmission is possible.

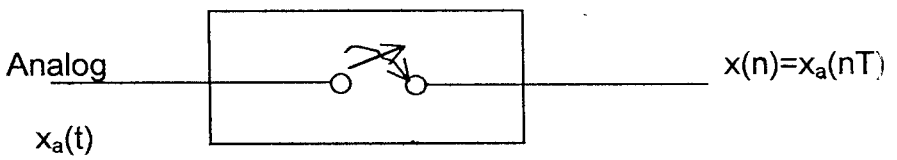


Fig. 2.1(a)

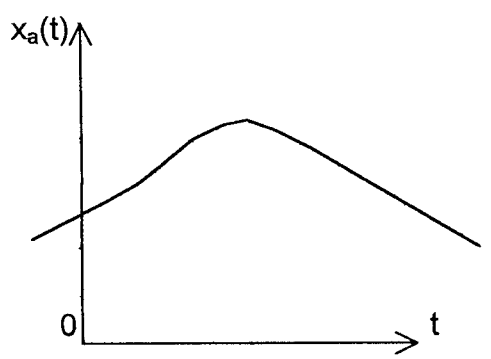


Fig.2.1(b)

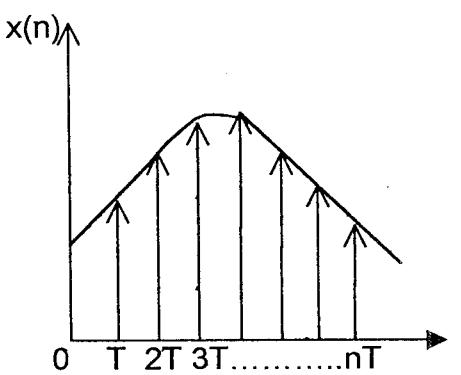
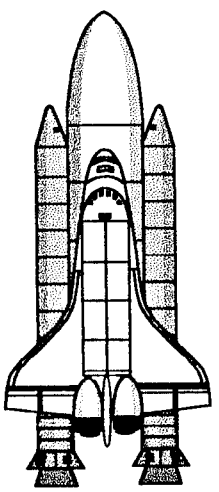


Fig. 2.1(c)



Frequency Shift Keyed Signals

3. FREQUENCY SHIFT KEYED SIGNALS

3.1. INTRODUCTION:

Digital signals can be modulated by several schemes such as Amplitude Shift Keying (ASK), Phase Shift Keying (PSK). Frequency Shift Keying (FSK) etc., Demodulation of digital modulated signals is similar to that of analog modulated signal. Here, for our study, we take into consideration the demodulation of digital signals by FSK.

When the datas are transmitted by varying the frequency, we have the case of Frequency Shift Keying (FSK). On-off keying is particularly susceptible to signal feeding. Thus it led to research of techniques, which would be less affected by feeding.

3.2 DESCRIPTION OF FSK SIGNALS

FSK was originally based on simple concept of using a telegraph signal to frequency modulate a career, with such frequency deviation as appeared to give a useful improvement in SNR in highest frequency of the signal (f_m).

However, the instantaneous frequency actually shifts quite rapidly between just 2 frequencies. One representing transmission of mark and the other, space. For wide durations, it becomes apparent that the 2 filter, one centered on mark frequency and the other on space frequency may provide a less noisy discrimination than the conventional between discrimination.

We could very well use amplitude, modulation with pulses ON corresponding to mark and OFF to space. However, such a system has Inherent disadvantage that there is no real indication for space. In addition, a system such as this would suffer from all usual of amplitude modulation, as a result of which it gives way to Frequency Shift keying at many a time.

Now–a-days, it has become very common to transmit digital signals for long distance. FSK modulation techniques is one of the modulation techniques used for modulating the digital signals. Normally they are known as binary FSK modulated signals, because the '1' state will be represented by one frequency say f_1 and the '0' state will be represented by another frequency f_2. Therefore, in the reception side it is necessary to demodulate the FSK modulated signals.

Let the f_m be information signal and f_c be carrier signal, then the FSK were obtained is. f_1KHz [OFF state or '0' level] and f_2KHz [ON state or I level], where $f_1=f_c-f_m$ and $f_2=f_c+f_m$. This is shown in Fig.3.1(a) and 3.1(b).

3.3 GENERATION OF FSK SAMPLES:-

A. SINE WAVE SAMPLES GENERATION

Analog version of FSK signals is converted into discrete by an analog to digital converter. The sampling rate is fixed as $f_s<2f_m$ (to avoid aliasing and overlapping of the signal). The presence of the signal is denoted as MARK and absence of the signal is denoted as SPACE. Mark and space frequencies used are 7 KHz and 9 KHz (i.e. 8K ± 1K) respectively. The center frequency of the carrier is 8 KHz. A sampling rate of 64 KHz is used.

Sampled version is given by

$$= V_m \sin 2\pi f_c \frac{n}{f_s}$$

$$= V_m \sin 2\pi f_c n T_s$$

T_s is the sampling interval

V_m amplitude of the FSK signal.

The 9 KHz and the 7 KHz samples are generated using the C language for simulation purpose, and they are stored in two different files. Programmes are added in software chapter.

B. FSK SAMPLE GENERATION:

Binary signal 1 corresponding to On state is represented by a 9 KHz signal and binary 0 corresponding to OFF state is represented by 7 KHz signal. In order to generate the FSK sample which is the mixture of 7 KHz and 9 KHz. The C language simulation is done, which converts the binary data to FSK samples. Also the graphical program is written to produce the monitor display of the FSK samples. The resultant values of the FSK samples are stored in the separate files, which is used for demodulation purpose. (This 'c' programme also added in the software chapter).

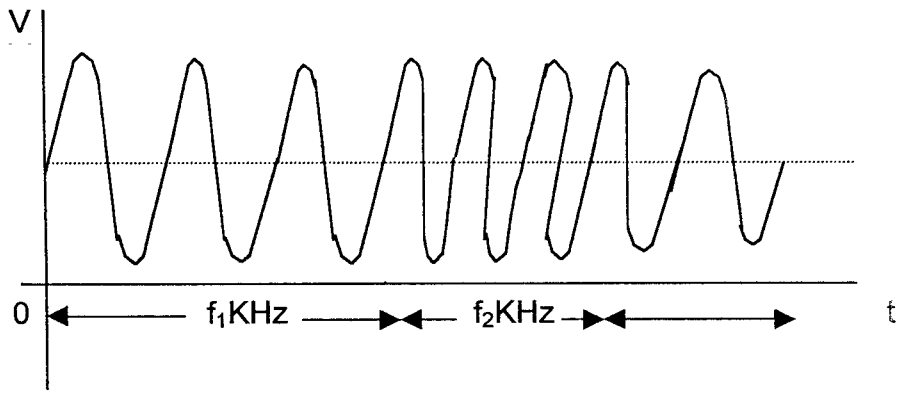


Fig. 3.1(a)

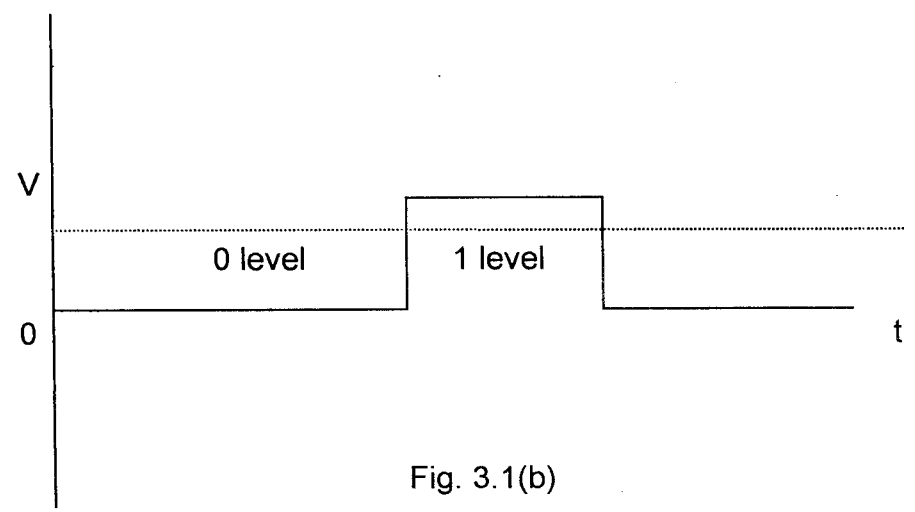
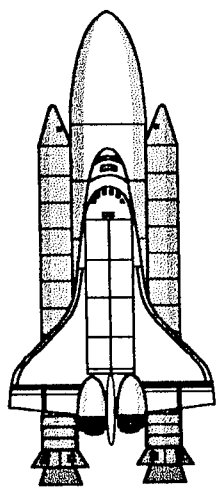


Fig. 3.1(b)



DSP Type Digital Phase Locked Loop

4. DSP TYPED DIGITAL PHASE LOCKED LOOP

4.1 INTRODUCTION:

With the development of modern digital technology, phase locked loops (PLL) are also becoming digitized. The digital signal processing (DSP) type digital PLL (DPLL) realised by Microcomputers and software deserves particular interest. This is because it can be used in time sharing operation and its characteristics can easily be controlled. So it is widely used in digital communication for synchronisation and signal conversion circuits.

4.2. DETAILS ABOUT DIGITAL PHASE LOCKED LOOP:

The DSP-DPLL has been derived from an analog PLL by directly realizing its operation with DSP methods.

The generation of distortion components due to non-linear characteristics of the phase deflection can be avoided in DSP type DPLL. The subtraction type phase detector is used, so the high frequency components are eliminated. The phase locked loop block diagram is shown in fig.4.1.

The input is digitised by analog to digital converter. The conversion is made according to Nyquist criteria.

If the input signal is

$$\aleph(K) = A \sin[2\pi f_i t + \theta(t)] + n(t)$$

f_i – input frequency

n(t) – gaussian noise component

θ(t) – input noise component

A - amplitude of the signal

The sampled version of the signal is

$$\aleph(K) = A \sin[2\pi f_i kt + \theta(k)] + n(k)$$

where t(k) = KT = K/f_s

θ(k) = $\theta[ct(k)]$

n(k) = n[t(k)]

f_s – sampling frequency

The figure 4.1 shows the digital phase locked loop

In the phase detector the input phase $\hat{\theta}(k)$ and the output phase is $\hat{\psi}(k)$ ie. output phase of Nω. So the output of phase detector is

$$\hat{\phi}(k) = \hat{\theta}(k) - \hat{\psi}(k)$$

In the case of linear PLL (DSP type) the characteristics of phase detector is linear from $-\pi$ to π. So distortion components are not generated, and moreover, neither are there any high frequency components. The harmonic suppression because of the subtraction type phase detector the excellent feature is that the output is only a dc component corresponding to the phase difference.

In this type the acquisition fine becomes shorter compared to nonlinear PLL. The SNR variance becomes nearly zero for high SNR. That's why subtraction type phase detector is widely used. The diagrams 4.2, 4.3, 4.4, 4.5 shows the difference characteristics between the linear and non linear PLL.

4.3 LOOP FILTER:

Loop filter in the PLL is used to remove the harmonics generated when the signal is sampled. The higher order FIR filter or IIR filter can be used. But here simple first order filter is sufficient because in the DSP DPLL the harmonics are very less. The loop filter diagram is shown in figure 4.7.

4.4 NUMBER COUNTING OSCILLATOR:

For generating a carrier sine wave, the usual methods are positive feedback oscillator or the phase locked loop synthesizer. In the positive feedback oscillator, at the desired frequency the circuit provides the right phase shift and gain as feedback. In the PLL synthesizer, there is a closed loop circuit which generated an error signal based on the difference between the input and output frequencies. That error signal is used to drive a **Voltage Controlled Oscillator (VCO)**. The VCO frequency is then divided by the programmable counter which provides the feedback necessary to lock the output frequency.

In the first method the sine wave is generated completely in the analog form. In the PLL synthesizer analog and digital forms are combined.

In the direct digital synthesizer the sine wave is generated completely in the digital form. The absolute frequency, phase tolerances, drift specifications and temperature stabilization are manageable. This is accomplished at lower cost and less complexity than while other synthesis techniques. In this the signal is generated in the form of a series of digital numbers and converted into analog form by a digital-to-analog converter.

The desired frequency is entered into the **Numerically Controlled Oscillator (NCO)** as a digital word and the NCO generates a digital sampled sine wave output that is locked to the crystal oscillator.

The functional block diagram of a DDS showing the five basic building blocks is shown in fig.4.8. (c).

The reference source is a stable crystal controlled oscillator used to synchronize the constituent parts at the oscillator. The accumulator is a device that converts frequency setting data into phase sampled which determine the magnitude of the synthesizer output waveform at a given sample time. When addressed with such data the sine look-up-table (ROM) transforms each phase sample into a digital amplitude sample at a sine wave that is converted into an analog signal by the DAC. The low pass filter attenuates the unwanted sampling components as well as other out of band spurious signals.

The frequency setting data which represent the desired frequency of the synthesized output signal are provided either by manually controlled switches or by a control unit such as a computer.

In fig.4.9 the input register accepts the digital data and presents it to the adder of the phase accumulator till a new frequency instruction is received. The adder adds the digital signals from the input register to the value of the accumulator register and updates the accumulator register with the most recent sum. The function of the accumulator register is to transfer the updated digital data from the output of the adder to the input at every clock pulse so as to make the module 2^n accumulator over flow periodically with a period determine by the frequency setting. This cycle of 2^n represents one cycle of the synthesized signal.

Phase increment = 2π (F_{out}/F_{REF}) rod,

Where F_{out} = Synthesized frequency

F_{REF} = Clock Frequency.

The phase increment is small for low synthesized frequencies and large for high frequencies.

When the synthesizer is set to a new frequency, the input register provides the adder with the data for a new phase increment at the next pulse of the clock at which time the period of the accumulator changes accordingly. The most significant bit of the accumulator output is a square wave with repetition rate of the synthesized signal. At each consecutive phase value generated by the phase accumulator, the ROM provides the corresponding digital number representing the magnitude of the synthesized waveform at that sample time.

The LPT attenuates all cut of band spurious signals generated in the process of frequency synthesis by the required amount. The number of phase accumulator bits determines the smallest frequency increment synthesized by using such a technique. It is equal to

$$\Delta f = \frac{F_{REF}}{2^N}$$

The highest synthesized frequency is limited by the DAC switching characteristics. At high frequencies, there are few samples per cycle of the synthesized signal with large amplitude differences between adjacent samples. To prevent the Digital to Analog Converter from introducing a high level harmonic distortion and generating spurious outputs, the settling time associated with switching from one sample to another should be very short compared to the sampling internal.

The advantages of the DDS techniques are numerous. The tuning time of DDS is an order of magnitude shorter than that of any other synthesizer. Any desired frequency resolution can be obtained at low cost, small volume, low DC power and lightweight by simply increasing the number of accumulator bits. It can generate frequency stable sine waves with a resolution down to 0.008Hz over its entire range. It is capable of changing the output frequency to a new value within one clock cycle without discontinuities in the waveform. Thus the phase of the synthesized signal is continuous as the synthesized frequency is changed which is essential in some application. The circuitry of DDS requires no alignment except for the DAC and LPF. This simplifies the manufacture of equipment and also reduces cost. The DDS can easily synthesize complex waveforms utilizing a combination of various modulation techniques such as FM, AM, FSK, BPSK, QPSK and MSK. The NCO modulator allows the designer to mix or multiply the sine wave with a digital pattern. The final output will depend both on the input digital pattern and the NCO sine wave.

The design limitation of DDS have poor DAC switching characteristics, the requirement of a reference source operating at a higher frequency than the synthesized frequency and the control of the aliased components. DDS is nosier than the other methods, but adequate spectral purity can be obtained if sufficient low pass filtering is used at the output. For an (n+1) bit word length (1 bit as sign bit), the worst case noise power (relative to the signal) is approximately $\sigma^2 = (2^n)^{-1}$ or $\sigma^2 = -6n$ dB. For each bit added to the word length the spectral purity improves by 6 dB.

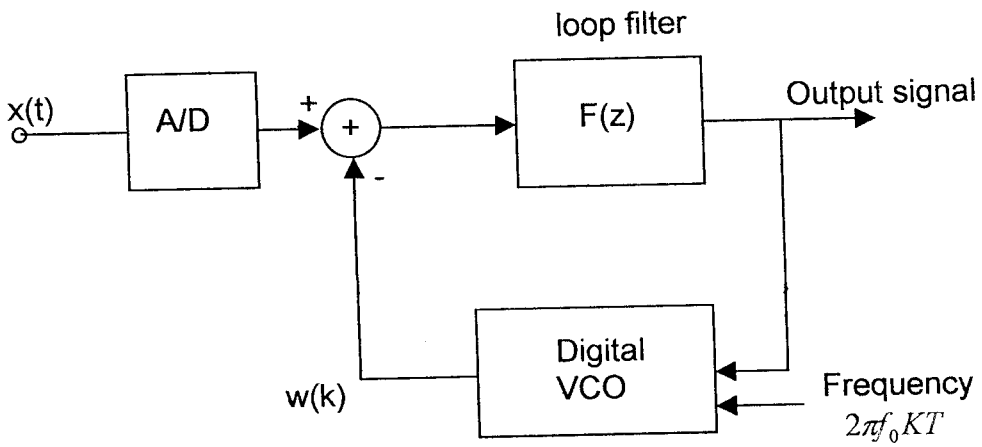
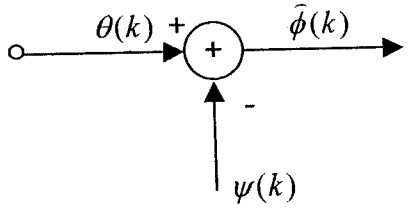


Fig. 4.1

COMPARISON OF MULTIPLIER PHASE DETECTOR AND SUBTRACTOR PHASE DETECTOR.

Subtractor



.dc component

(ϕ Linear)

Fig.4.2(a)

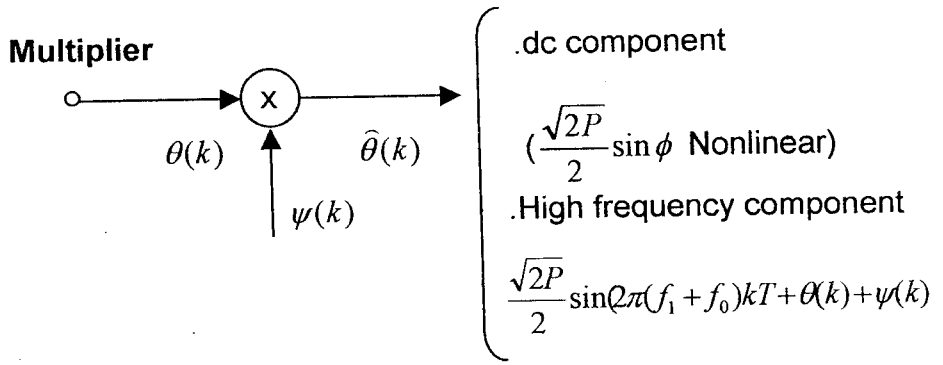


Fig. 4.2(b)

PHASE DEFLECTION CHARACTERISTICS

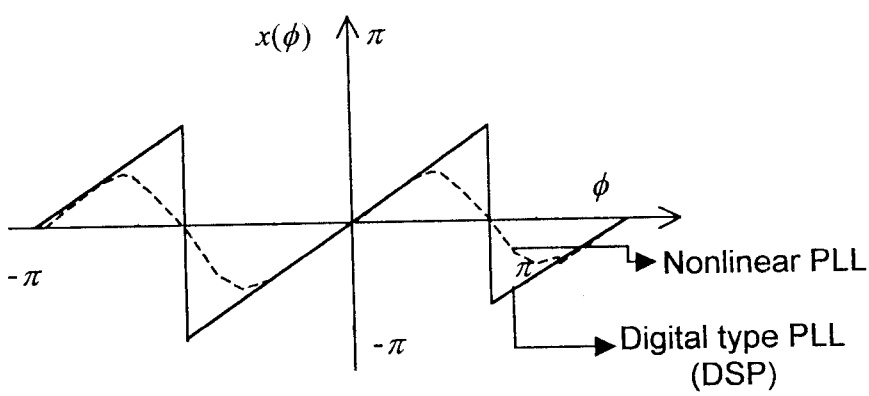


Fig. 4.2(c)

LOOP GAIN VS ACQUISITION TIME CHARACTERISTICS

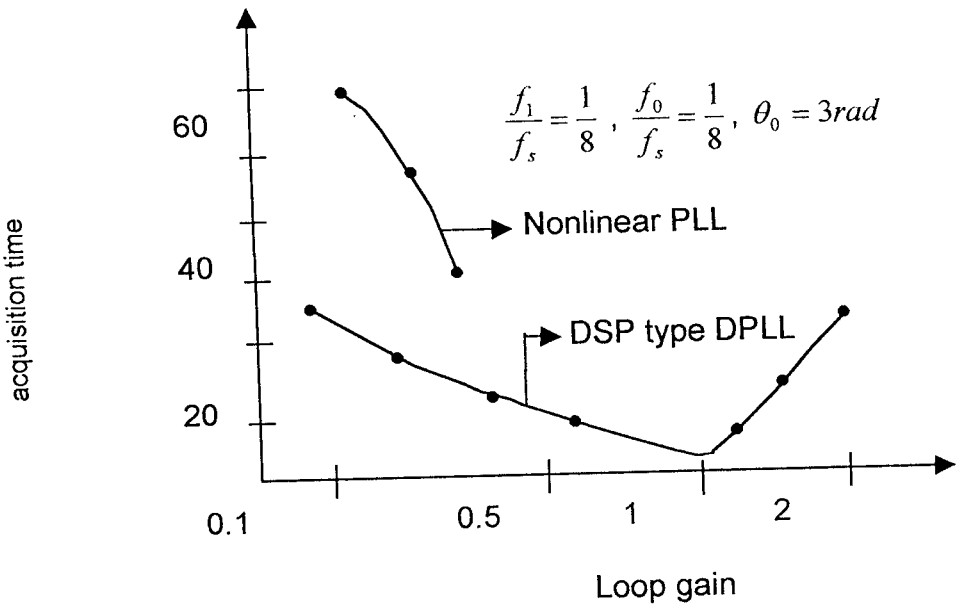


Fig. 4.3

INITIAL PHASE ERROR Vs ACQUISITION TIME:

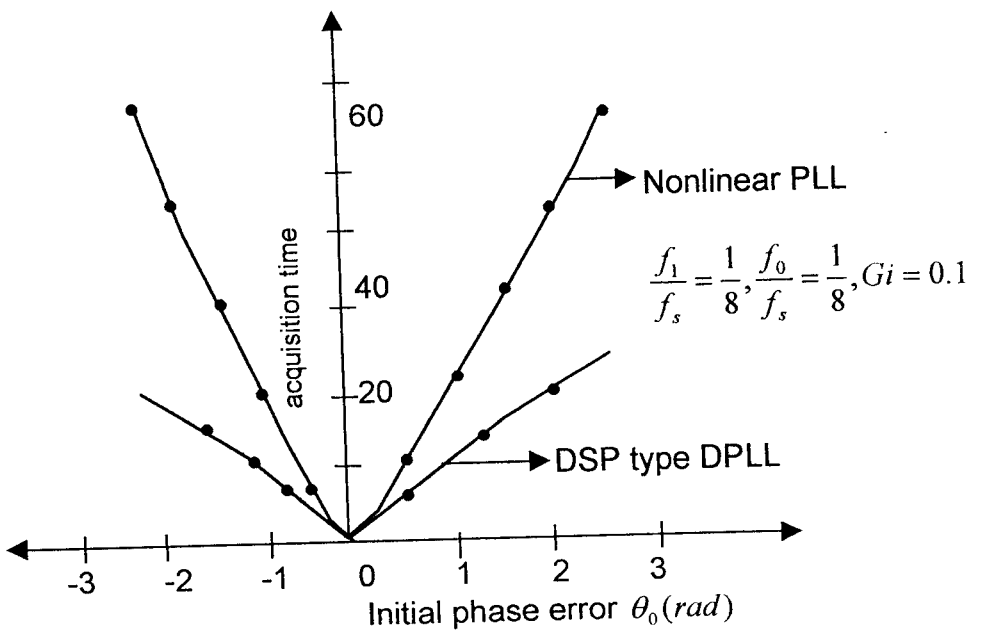


Fig. 4.4

EXAMPLE OF ACQUISITION:

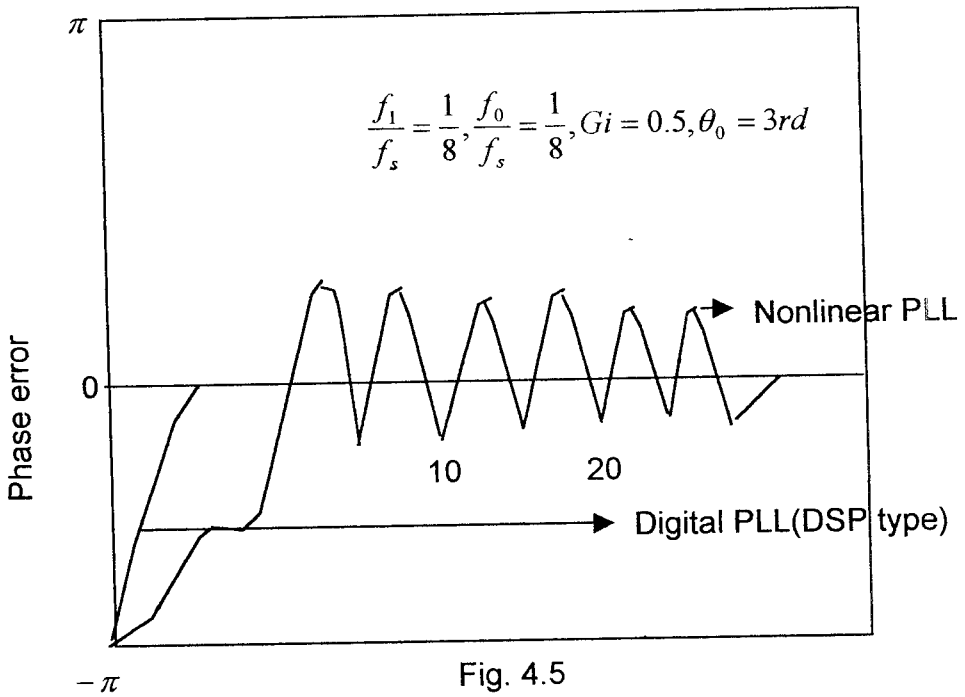


Fig. 4.5

CHARACTERISTICS OF PHASE ERROR VARIANCE

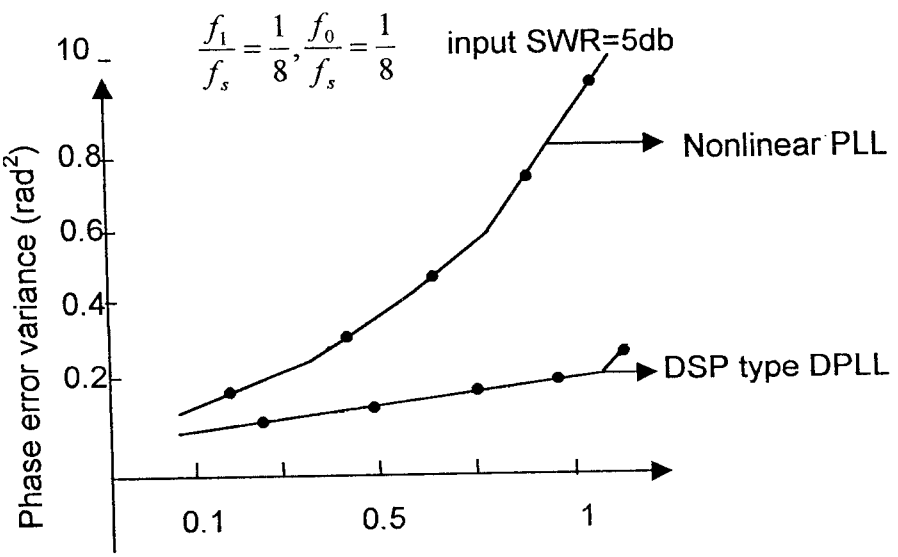
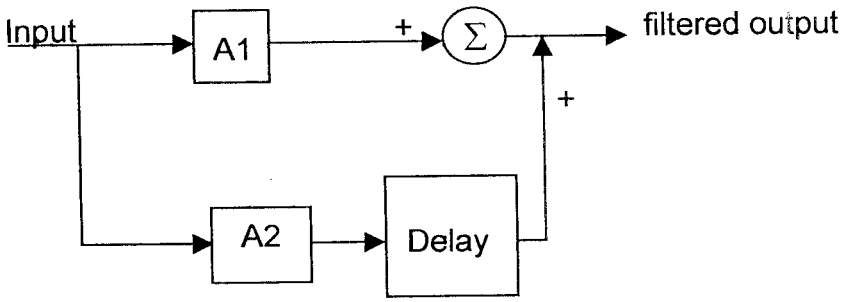


Fig. 4.6



A1, A2 gain

A1=A2=1/2

Fig. 4.7

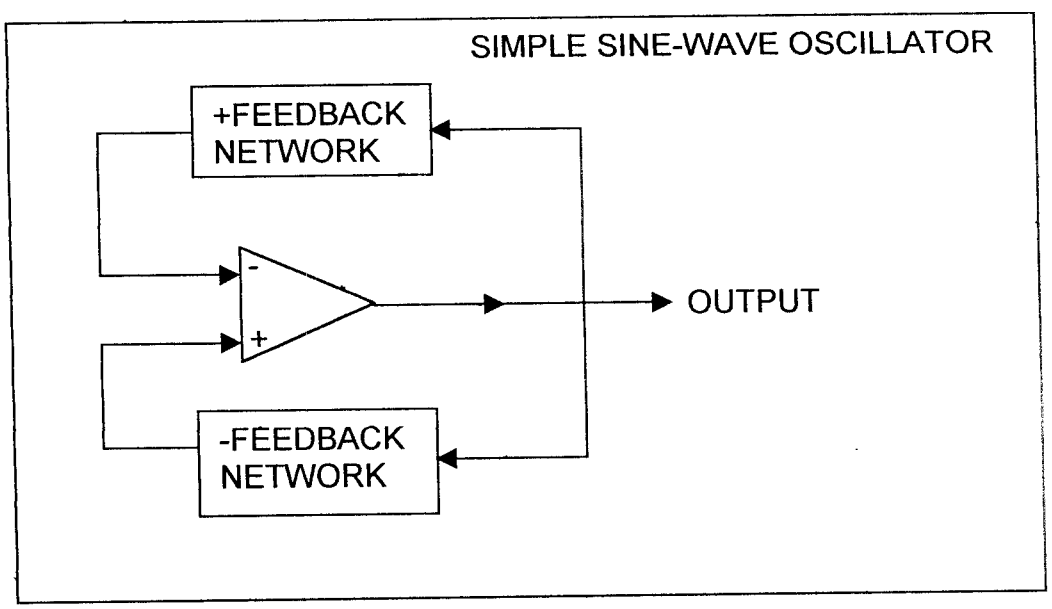


Fig. 4.8.(a). POSITIVE FEEDBACK OSCILLATOR

[from Gene Pikus (1995)]

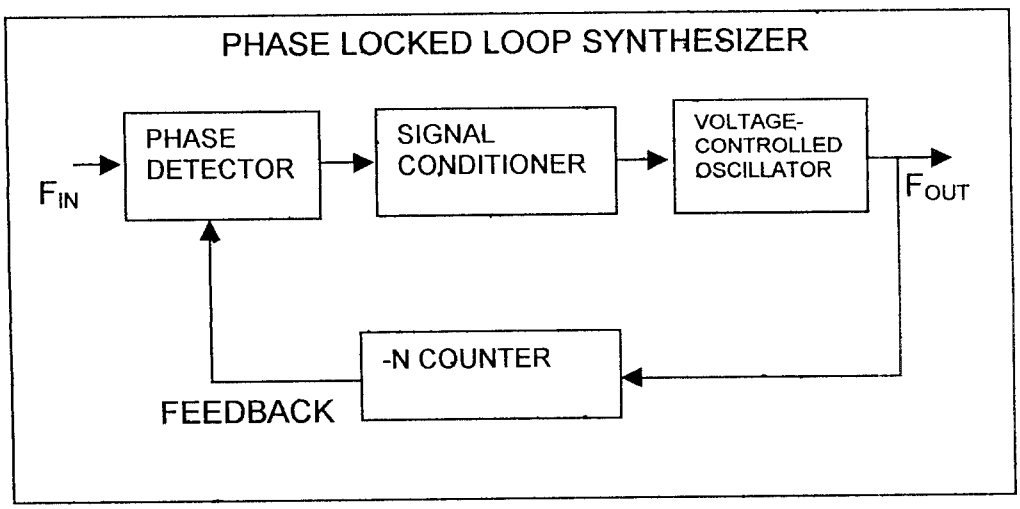


Fig. 4.8.(b). PLL SYNTHESIZER

[From Gene Pikus (1995)]

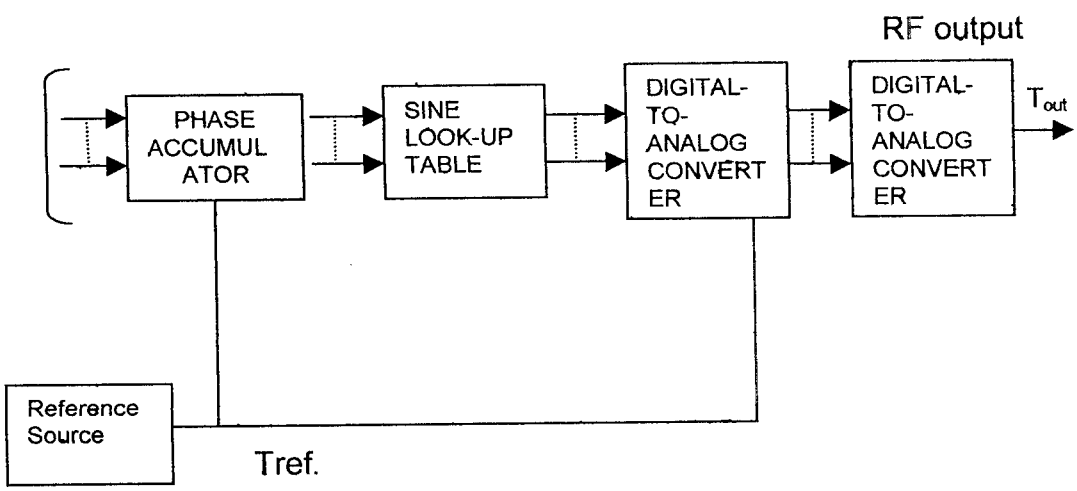


Fig.4.8.(c) DIRECT DIGITAL SYNTHESIZER;
SIMPLIFIED BLOCK DIAGRAM
[From manasseritsch (1981)]

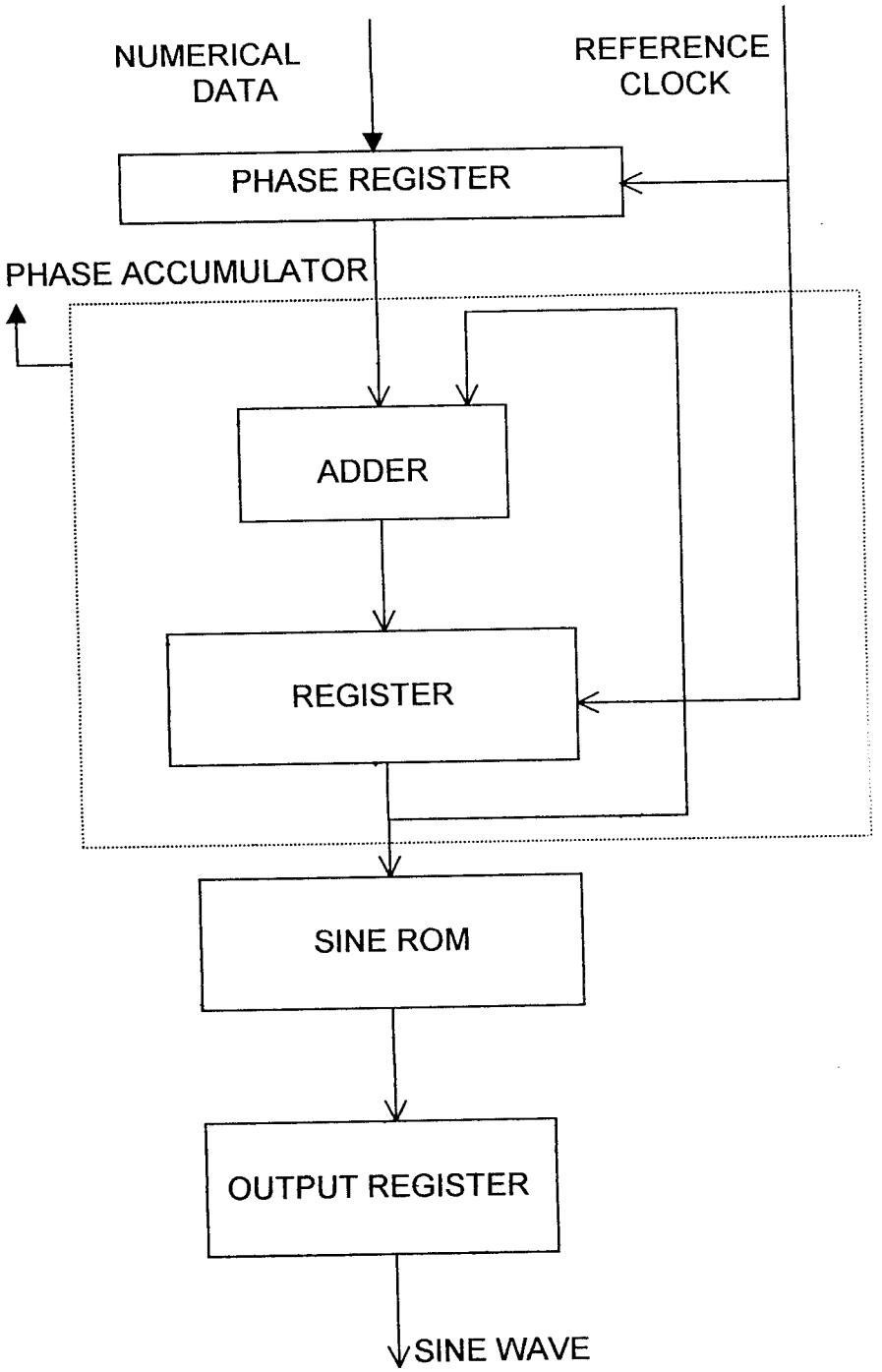
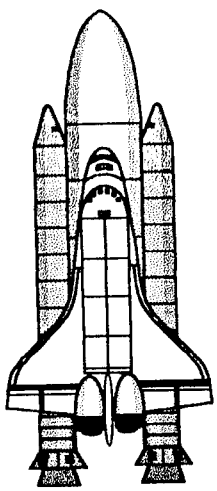


Fig. 4.9 DDS BLOCK DIAGRAM



DSP Processor for DPLL Implementation

5. DSP PROCESSOR FOR DPLL IMPLEMENTATION

5.1. INTRODUCTION:

In real time application everything should be very fast. If we are giving any signal to the input of any systems that should be immediately converted as desired (within few micro or nano sec). That's why we are using DSP processor for our application.

The DSP type DPLL can be implemented in microcomputers, microprocessors and software deserves particular interest. So it can be used in time sharing operation and the characteristics can be easily be controlled.

5.2. DSP PROCESSORS:

The DSP processors will execute one instruction and fetch the next instruction in a single T state. They can receive and transmit data via the serial codes. There are many DSP processors available for example.

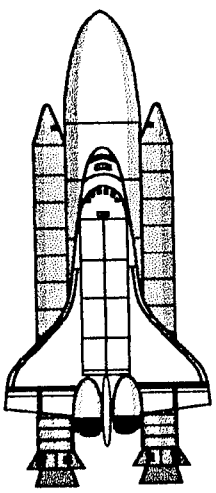
1. TMS 320 C40
2. ADSP 2101
3. ADSP 21060

ADSP 2101 is used for this project (Analog devices DSP Micro Computer), because the DPLL used is in the on board circuitry of the tele command communication network. The on board memory is so fast, the ADSP 2101 can fetch and execute the operand with very high speed.

And also the cycle time is very less and memory available is sufficient.

That's why we are very much interested in ADSP 2101 processor to implement digital PLL.

The C source code (or) the instruction sets of ADSP 2101 can be written to develop the implementation of digital PLL algorithm.



Demodulation Software Development

6. DEMODULATION SOFTWARE DEVELOPMENT

6.1. GENERATION OF SINE WAVE:

This is a software written in "C" language to generate a sine wave of any frequency with the corresponding sampling rate. The desired V_{max}, frequency, sampling rate can be entered. This program also includes another program which helps in drawing the sine waveform. The values of the sine wave generated are stored in a separate file. The frequency values are generated by the formula.

$$K = \text{Sine}\ (2 \times 3.14 \times freq \times n \times T)\ ;$$

$$T = \frac{1}{Samp.freq.}$$

Here 'n' is incremented from O to a value depending on the sampling frequency. If the sampling frequency is 50 KHz, then 'n' is incremented from 0 to 50. 'K' is multiplied by 100 to magnify the values obtained.

[Sine wave generation program with Graph is added in Software Chapter]

6.2 GENERATION OF FSK SAMPLED SIGNALS:

This program generates the sampled values of FSK. The values can be entered in terms of '0' and 'I' representing 'space' and 'mark' respectively. If the entering of values is to be stopped then '50' is to be entered.

Another program is written in order to draw the graph. The values are stores in separate files.

The values 'O' and 'I' may represent, say for example, 7 KHz and 9 KHz. The FSK signal consists of sequence of 7 KHz and 9 KHz, depends on the signal.

[FSK sample Generation with graph is added in Software Chapter]

6.3 DESCRIPTION OF THE BLOCK DIAGRAM

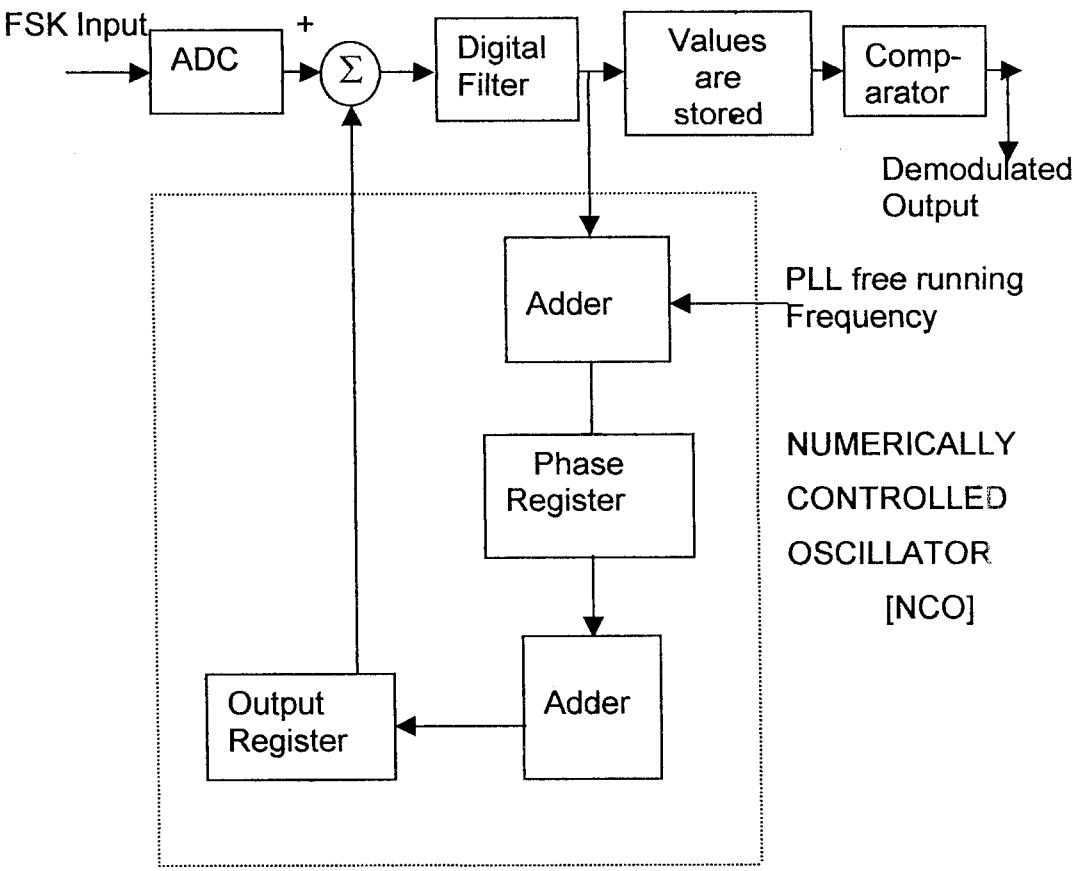
The FSK sampled input which is analog in nature is to be converted to digital. This is done by making use of a Analog to Digital converter (ADC). This analog to digital converter can be of any type:

The sampled digital output is fed to the phase detector. The phase detector is a device that detects phase error of the i/p signal and that of the NCO output. This is then fed to the digital filter. The digital filter is an ordinary loop filter of first order. Higher order filters are not necessary because even a first order filter is capable of removing the harmonics generated in this process.

The filtered output, which is free of harmonics, is fed to the Numerically Controlled Oscillator (NCO) and also to the device used for squaring. The numerically controlled oscillator part consists of adders, phase register, and output registers. The adder performs the task of adding the harmonics – free output and the PLL free running frequency. This sum is stored in a register. This value is added with the previous values till a full sine wave is obtained. It is then fed to the phase detector. The function of phase detector was described before.

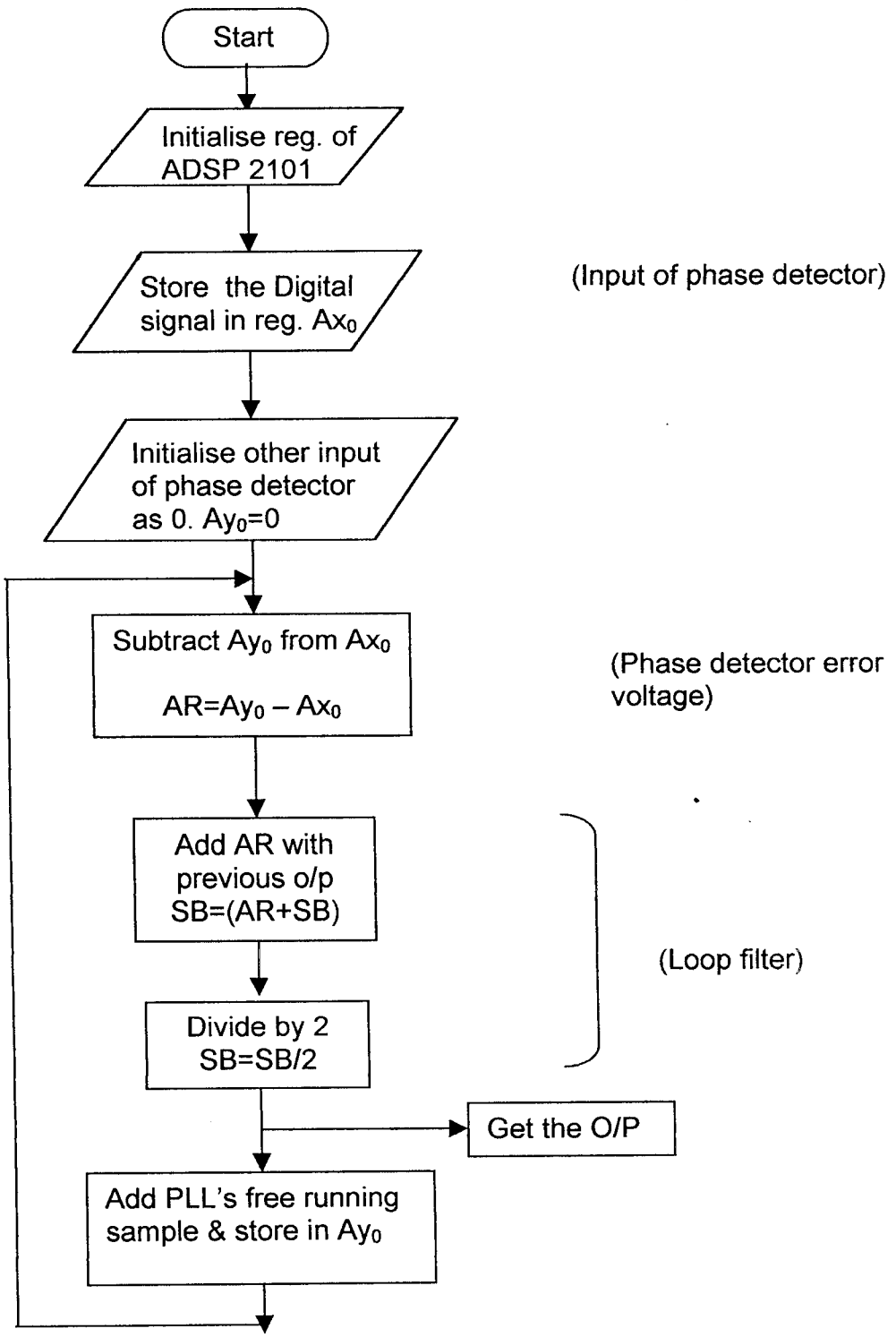
As mentioned earlier, the output of the digital filter is also fed to the squaring device. The squaring device will suppress the harmonics with very little magnitude. This value is fed to the comparator. The comparator compares the values and gives the output. Comparison is done by taking three values at a time and adding it. If the summed up value is greater than threshold, it is considered as 'I' else as 'O'.

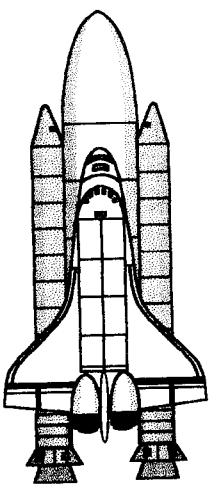
PHASE DETECTOR



FSK Centre Freq.	: 8KHz
Deviation	: 1KHz
Amplitude	: 1 Volt
Data rate	: 1 Kbps
Sampling rate	: 64 KHz.

This whole procedure is described in the flowchart given below:





Result and Conclusion

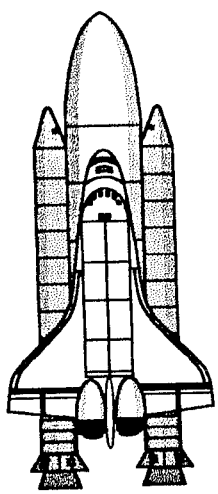
7. RESULTS AND CONCLUSION

The FSK signal is demodulated using DPLL algorithm which is implemented in ADSP 2101 simulator. The demodulated output is obtained.

The main advantages of using this ADSP 2101 processor is that we can use this processor in on-board circuitry for real time applications. This is because it has a faster response time than other processors.

The obtained results, (ie.) demodulated data, suffers deviation for the binary datas 0 and 1. In this case, by setting proper threshold level for the comparator, the result can be obtained.

The deviation here, is due to the usage of ordinary first order digital filter. The deviation causes error in real time applications. Hence it is more advantages of a higher order filter is used.



Appendix

APPENDIX – A

ADSP 2101 PROCESSOR DETAILS

FEATURES:

Complete 50 MHz DSP Microcomputer ADSP-2100 Code & Function Compatible 2K words of program memory RAM 1K word of data memory RAM separate program and data bases on-chip dual purpose program memory for Both Instruction and Data storage.

Three Independent Computational Units; ALU, Multiplier / Accumulator and Barrel Shifter. Two Independent Data Address Generators. Powerful program sequencer, Zero Overhead Looping conditional Arithmetic Instruction Execution, Two Double-Buffered Serial Ports with Companding Hardware and Automatic Data Buffering Programmable Interval Timer.

Programmable Wait State Generation. Automatic Booting from Byte-Wide External memory. eg. EPROM.

Provisions for Multiprecision Computation and Saturation Logic. Single-Cycle Instruction Execution. Multifunctional Instructions. Three Edge or Level-Sensitive External Interrupts 80 ns Cycle Time. 30 mW Maximum Power Dissipation in Standby Mode 68-Pin PGA and 68-Lead PLCC.

GENERAL DESCRIPTION.

The ADSP-2101 is a single-chip microcomputer optimized for digital signal processing (DSP) and other high-speed numeric processing applications. Its instruction set is a fully compatible superset of the ADSP-2100 instruction set. It combines the complete ADSP-2100 architecture (three computational units, data address generators and a program sequence) with two serial ports, a programmable timer, extensive interrupt capabilities and on-board program and data memory RAM. The ADSP-2101 has 1K words of (16-bit) data memory RAM and 2 K words of (24-bit) program memory RAM on chip. The ADSP-2101 surpasses other single-chip DSP microcomputers in both performance and case of design and development.

Fabricated in a high-speed 1.0 micron double-layer metal CMOS process, the ADSP-2101 operates at 50 MHz. Every instruction executes in a single cycle. Fabrication in CMOS results in low power requirements. The ADSP-2101 dissipates less than 1W under all conditions and no more than 80 mW under standby conditions.

The ADSP-2101's flexible architecture and comprehensive instruction set support a high degree of operational parallelism. In one cycle the ADSP-2101 can:

- generate the next program address
- fetch the next instruction
- perform one or two data moves
- update one or two data address pointers
- perform a computational operation
- receive and transmit data via the two serial ports.

DEVELOPMENT SYSTEM:

The ADSP-2101 is supported by a complete set of tools for software and hardware system development. The cross-software system is a set of modules. The System Builder provides a high-level method for defining the architecture of systems under development. The Assembler produces object code and the Linker combines object modules and library calls into an executable file. The simulator provides an interactive instruction-level simulation with a reconfigurable user interface. A PROM Splitter generates PROM burner compatible files. The C Compiler generates ADSP-2101 assembly source code. An Emulator aids in the hardware debugging of ADSP-2101 systems.

ARCHITECTURE OVERVIEW:

Figure 1. is an overall block diagram of the ADSP-2101. For compatibility with the ADSP-2100 processor, the additional features of the ADSP-2101 appear in the form of new mode controls, new processor registers and a group of memory mapped control registers residing between data memory addresses.

H#3FE0 and H#3FFF.

The processor contains three independent computational units; the ALU, the multiplier/accumulator (MAC) and the shifter. The computational units process 16-bit data directly and have provisions to support multiprecision computations. The ALU performs a standard set of arithmetic and logic operations; division primitives are also supported. The MAC performs single-cycle multiply, multiply/add and multiply/subtract operations. The shifter performs logical and arithmetic shifts, normalization, denormalization, and derive exponent operations. The shifter can be used to efficiently implement numeric format control including multiword floating-point representations.

The internal result (R) bus directly connects the computational units so that the output of any unit may be the input of any unit on the next cycle.

A powerful program sequence and two dedicated data address generators ensure efficient use of these computational units. The sequence supports conditional jumps, subroutine calls and returns in a single cycle. With internal loop counters and loop stacks, the ADSP-2101 executes looped code with zero overhead; no explicit jump instructions are required to maintain the loop.

The data address generators (DAGs) handle address pointer updates. Each DAG keeps track of four address pointers. Whenever the pointer is used to access data (indirect addressing), it is post-modified by the value of a specified modify register. A length value may be associated with each pointer to implement automatic modulo addressing for circular buffers. With two independent DAGs, the processor can generate two addresses simultaneously for dual operand fetches. The circular buffering feature is also used by the serial ports for automatic data transfers; these are described in the section on serial ports.

Efficient data transfer is achieved with the use of five internal buses.

- Program Memory Address (PMA) Bus
- Program Memory Data (PMD) Bus
- Data Memory Address (DMA) Bus
- Data Memory Data (DMD) Bus
- Result (R) Bus

The two address buses (PMA and DMA) share a single external address bus, and the two data buses (PMD and DMD) share a single external data bus. The BMS, DMS and PMS signals indicate which memory space the external buses are being used for.

As in the ADSP-2100 program memory can store both instructions and data, permitting the ADSP-2101 to fetch two operands in a single cycle, one from program memory and one from data memory. Because the on-board program memory is so fast, the ADSP-2101 can fetch an operand from program memory and the next instruction in the same cycle. This eliminates the need for the cache memory found on the ADSP-2100, as well as any overhead cycles that were associated with initial loading of the cache.

The memory interface supports slow memories and memory-mapped peripherals with programmable wait state generation. External devices can gain control of buses with bus request/grant signals (BR and BG). One execution mode allows the ADSP-2101 to continue running while the buses are granted to another master as long as an external memory operation is not required. The other execution mode requires the processor to halt while buses are granted.

The ADSP-2101 instruction set provides flexible data moves and multifunctional (one or two data moves with a computation) instructions. Every instruction can be executed in a single processor cycle. The ADSP-2101 assembly language uses an algebraic syntax for case of coding and readability. A comprehensive set of development tools supports program development.

INSTRUCTION SET DESCRIPTION:

The ADSP-2101 assembly language, like the ADSP-2100, uses an algebraic syntax for case of coding and readability. The sources and destinations of computations and data movements are written explicitly in each assembly statement, eliminating cryptic assembler mnemonics. Every instruction assembles into a single 24-bit word and executes in a single cycle. The instructions encompass a wide variety of instruction types along with a high degree of operational parallelism. There are five basic categories of instructions, computational instructions, multifunction instructions, program flow control instructions and miscellaneous instructions. Each of these instruction types is described briefly. The complete instruction set is summarized at the end of this section. The ADSP-2101 User's Manual gives an overview and the ADSP-2101 cross-software Manual contains a complete reference to the instruction set.

ADSP-2100 COMPATIBILITY:

The ADSP-2101 instruction set is a superset of the ADSP-2100 instruction set. The ADSP-2101 is source and object code compatible with the ADSP-2100. An ADSP-2100 program may need to be relocated to utilize internal memory and conform to the ADSP-2101's interrupt vector and reset vector placement.

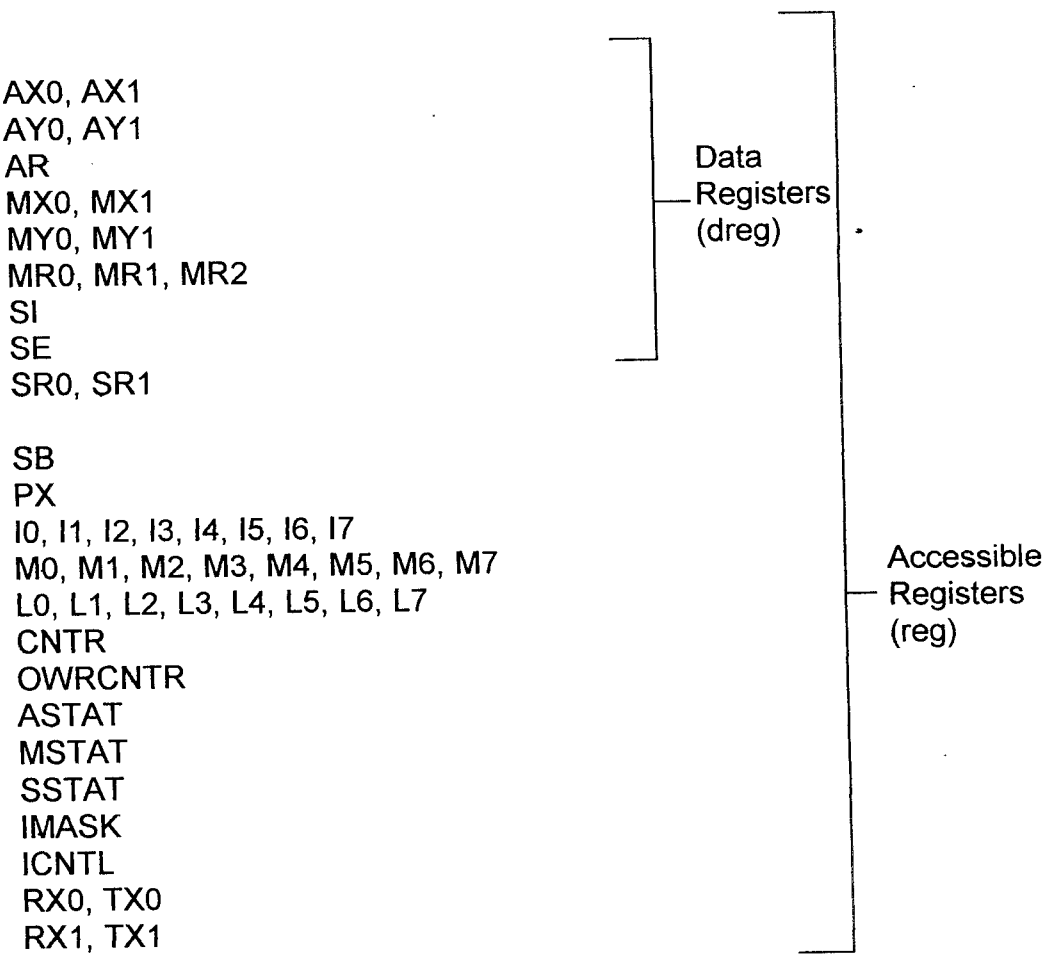
DATA MOVE INSTRUCTIONS:

The set of registers are denoted as reg in the instruction set summary given in Table-A subset of the reg group associated with the computational units, which generally hold data as opposed to address or status information, are denoted as dreg. Memory-mapped control registers are accessed as data memory locations, not as registers.

The data move instructions include transfers between internal registers, between data memories and internal registers, between program memories and internal registers, and immediate value loading of registers and data memories. The content of every reg can also be copied to any other reg.

Two addressing modes are supported for data memory transfers; direct addressing and indirect addressing. In direct addressing, the memory address is supplied from the instruction word. In indirect addressing, one of the data address generators provides the address. Using direct addressing, the contents of a data memory location can be written and read by any reg. Using indirect addressing, the contents of a data memory location can only be written and read by a dreg; Immediate

data load to data memory is permitted with indirect addressing. Only the indirect addressing mode is supported for program memory data transfers and contents of a program memory location can be read and written to any dreg.



COMPUTATIONAL INSTRUCTIONS:

There are three types of operations associated with the computational units: ALU operations, MAC operations, and shifter operations. With few exceptions, all these computational instructions can be made conditional. (The permissible conditions are specified in Table-II.) Each computational unit has a set of input registers and output registers. A list of permissible input operands and result registers for each of the unit is given in Table.

ALU

Source for X Input(xop)	Source for Y Input (yop)	Destination for Output Port R
AX0, AX1 AR MR0, MR1, MR2 SR0, SR1	AY0, AY1 AF	AR AF

MAC

Source for X Input(xop)	Source for Y Input (yop)	Destination for Output Port R
MX0, MX1 AR MR0, MR1, MR2 SR0, SR1	MY0, MY1 MF	MR(MR2, MR1, MR0) MF

Shifter

Source for X Input(xop)	Destination for Shifter Output.
SI AR MR0, MR1, MR2	SR(SR1, SR0)

Computational Input / Output Registers.

MULTIFUNCTION INSTRUCTIONS:

Multifunction instructions execute one computational operation with one or two data moves. All of the multifunction instructions utilize various combinations of the computational and data move operations described above. Since the instruction word is only 24 bit wide, only certain combinations are valid. In general, the following rules are followed.

- Computation must be unconditional
- Any memory transfer must use the indirect addressing mode
- Data move operations can only use data registers (dregs).

LINKER

INTRODUCTION:

The ADSP-21xx linker generates an executable program by linking together separately-assembled modules. The linker's primary output is a memory image file for the program which has the filename extension .EXE. This file is loaded into the ADSP-21xx simulators and emulators for debugging. Once the program is fully debugged. Once the program is fully debugged, the PROM splitter tool is used to generate PROM burner files from the memory image file.

The assembler processes each source code module and outputs an object file (.OBJ), a code file (.CDE), and an initialization file (.INT). The object file contains memory allocation and symbol information, while the code file contains ADSP-21xx opcodes with unresolved symbols marked. The initialization file contains information regarding data variables and buffers. The initialization data must be supplied by data files named with the assembler's .INIT directive. The linker reads data from these files and incorporates it into the .EXE memory image file. Changes in initialization data only requires relinking. Figure-3, shows the files input and output by the linker.

The linker scans each assembled module and resolves global / external symbol references between modules. It assigns ("allocates") addresses to relocatable code and data fragments. The linker also reads the .ACH architecture description file in order to construct the system memory map and allocate code and data to it. You must identify your architecture file for the linker with the a invocation line switch.

The linker can generate three different files, the .EXE memory image file is always created – this is the executable program – and contains the actual opcodes and data to be stored in memory. The optional .MAP map listing file summarizes information regarding the linked program. The optional .SYM symbol table file lists all symbols encountered by the linker, their absolute values and their scope of reference. This file is also used by the ADSP-21xx simulators and emulators.

The initialization data files (.DAT) re not explicitly named in the linker invocation since they are specified (with the .INIT directive) in the source code files. The data files are incorporated by the linker. When changes are made in the data files, simply relink to incorporate the new data.

RUNNING THE LINKER:

The linker is invoked this way, listing the files to be processed:

LD21 file1 [file2...] [-switch...]

For example:

ld21 main sub1 sub2 –a archfile

Each input file must contain only one module. If the files are not in the current directory of your operating system, a path must be given with each filename to enable the linker to find the directory where it is stored. The filenames must identify the assembler-output files with no extension (ie. CDE, .OBJ, .INT). The linker-output files are given the default filename 210X. Rename of these files are possible by using the –Entrepreneurs switch (see "Linker Switch Options" below).

LD21 –I file_all [-switch...]

In this case the linker reads the indirect list file file_all, which must be a simple text file with one path / filename per line.

Other optional switches control various aspects of linker operation. They may be entered in either upper or lower-case. Multiple switches must be separated by at least one space.

The linker allocates memory to modules according to the order in which the input files are listed. For modules which contain circular data buffer declarations, changing the order of input files may determine whether or not a program can be successfully linked in the available memory space. This leads to the following guideline:

Modules containing circular buffers of different sizes should be listed according to descending buffer size.

This forces the linker to allocate memory to the larger circular buffers first (which have greater restrictions on allowable base addresses).

For help,

LD21 – help

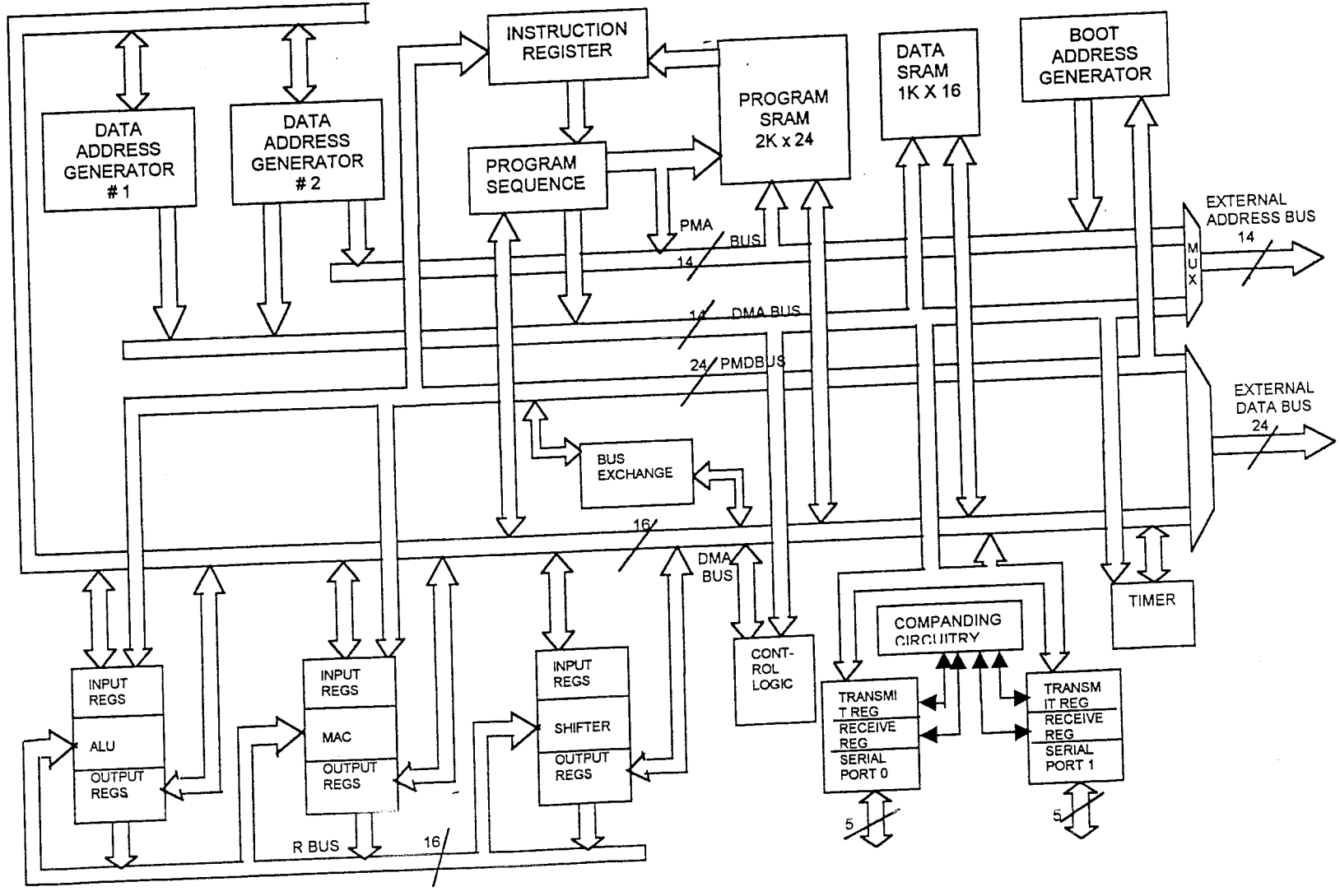
This will show you how the command must be entered and will display a list of the available switches. The help switch works with all the development software tools.

The assembled source code is generated by the ADSP-21xx C Compiler, the linker must be invoked with its –c switch.

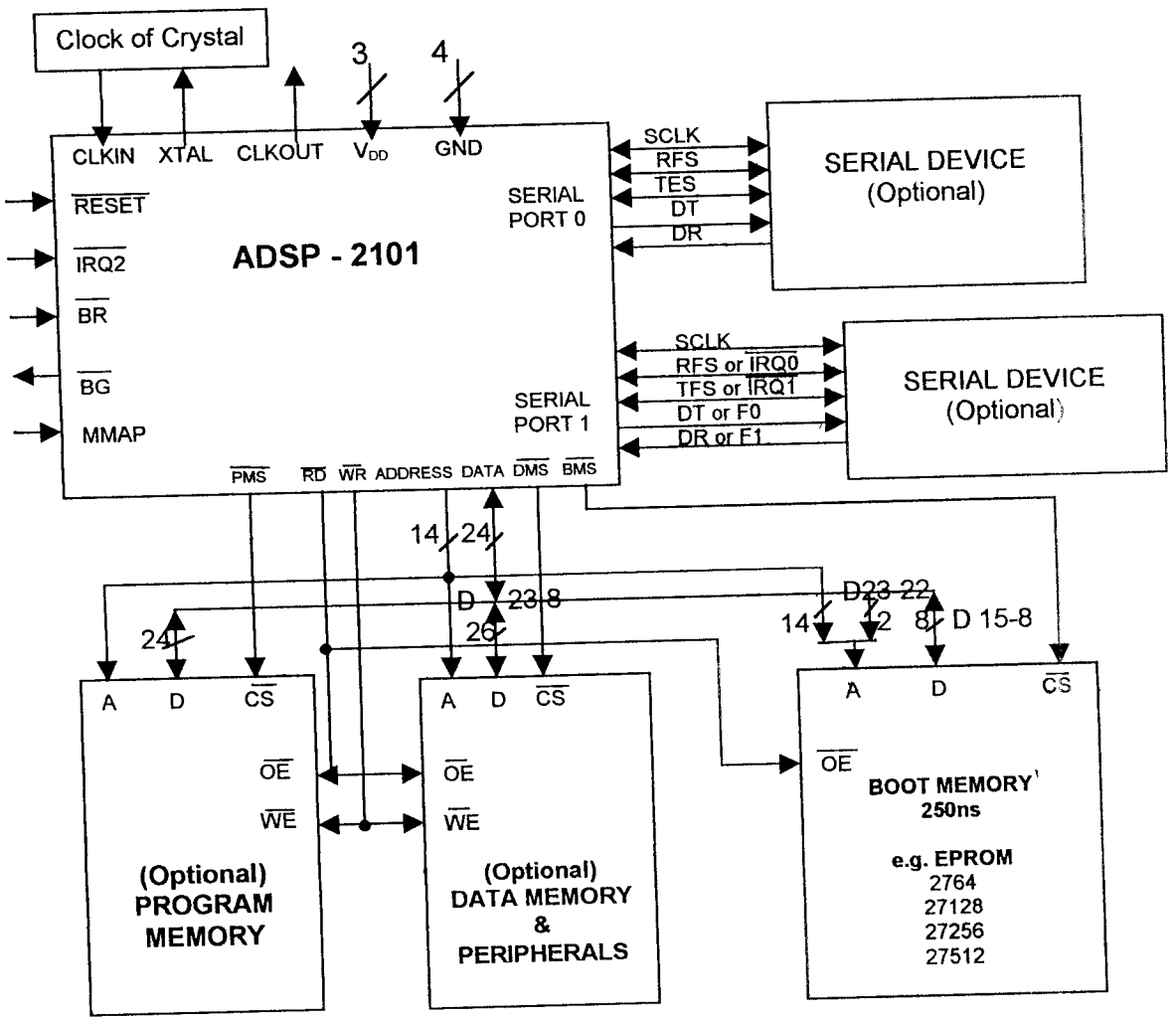
LINKER SWITCH OPTIONS:

The linker switches are listed below in Table, some require arguments as shown:

Switch	Effect
-a archfile	Architecture description file archfile .ACH read by llinker
-c	Runtime stack created for compiled C programs (in DM)
-dir directory;...	Specify directories to search for library routines
-dryrun	Quick run to test for link errors (no .EXE file generated)
-e executable	Output files given filename executable (default is 210X)
-g	.SYM symbol table file generated
-i file_all	Files listed in indirect file file_all are linked
-lib	ADSP-21xx Runtime C Library linked (use only with –c)
-p	Assign library routines to boot pages where called
-pmstack	C stack moved to program memory (only with –c)
-rom	ROM version of Runtime C Library used (use only with –c)
-user fastlibr	Search fast library file generated by LIB21 Library builder utility.
-x	.MAP listing file generated.



ADSP – 2101 BLOCK DIAGRAM



ADSP – 2101 Basic System Configuration

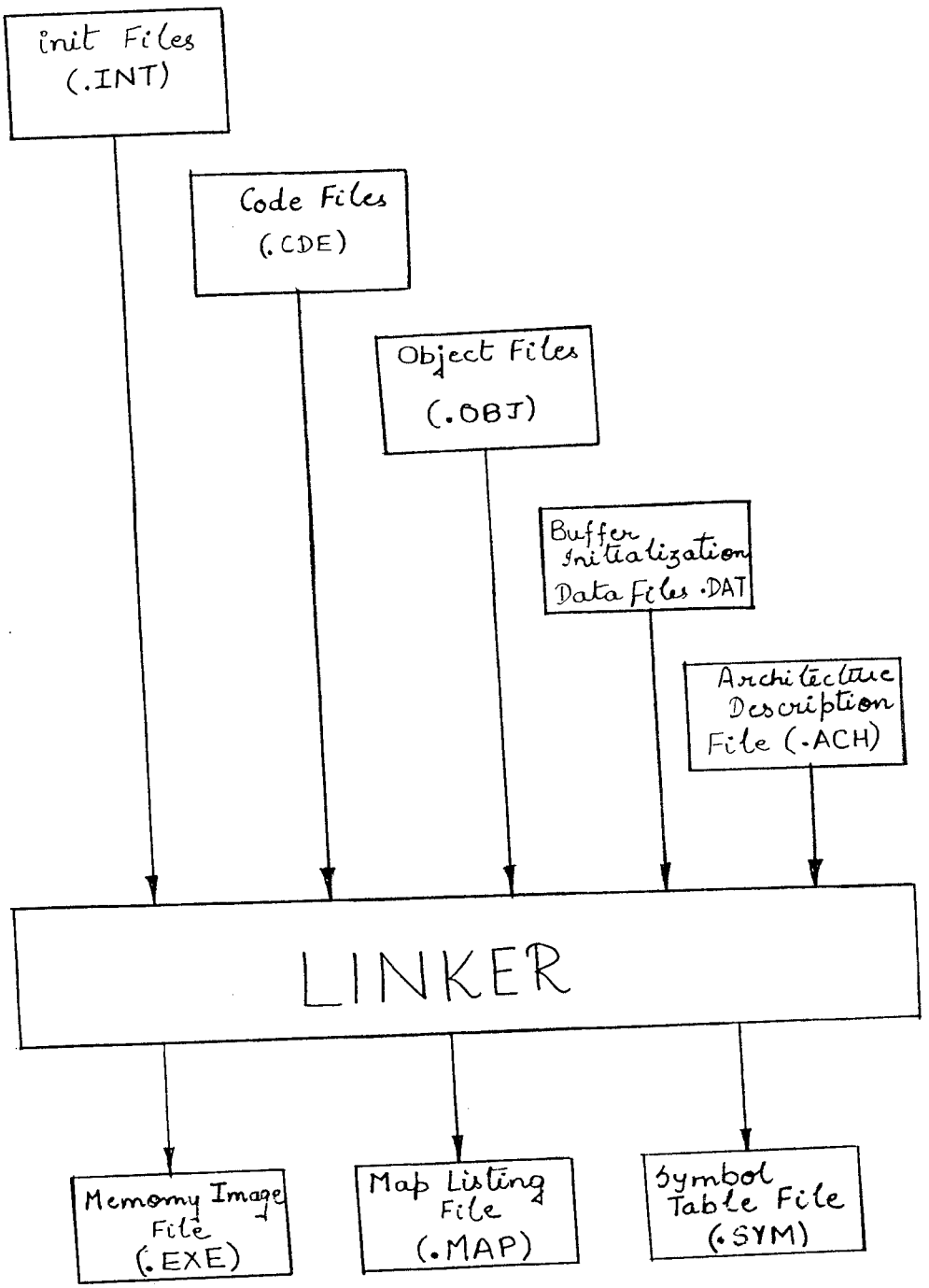

init Files
(.INT)
Code Files
(.CDE)
Object Files
(.OBJ)
Buffer
Initialization
Data Files .DAT
Architecture
Description
File (.ACH)
LINKER
Memomy Image
File
(.EXE)
Map Listing
File
(.MAP)
Symbol
Table File
(.SYM)

FIG - A.3

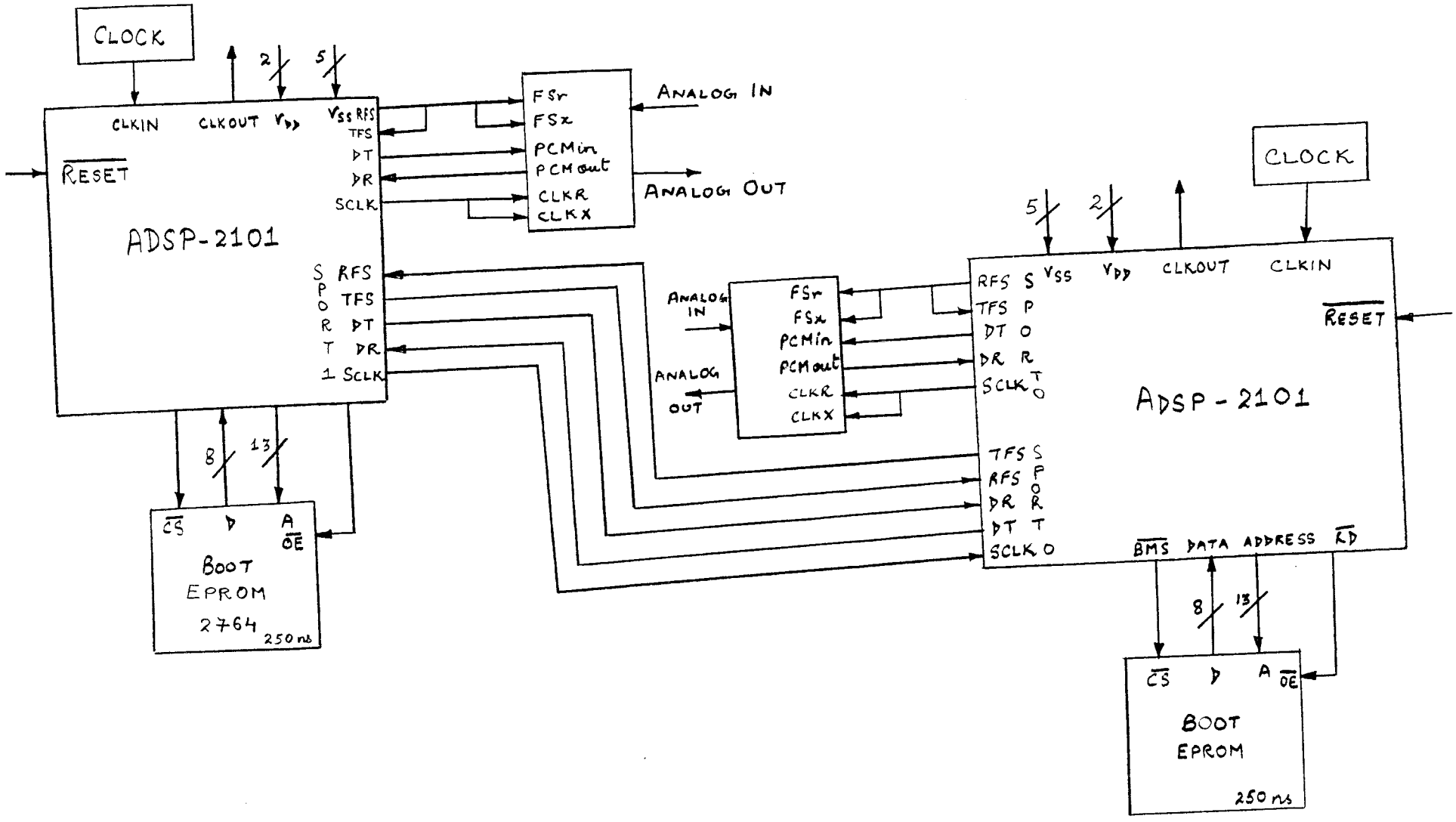

CLOCK
2
5
CLKIN
CLKOUT
VDD
VSS
RFS
TFS
DT
DR
SCLK
RESET
ADSP-2101
SPORT1
FSr
FSx
PCMin
PCMout
CLKR
CLKX
ANALOG IN
ANALOG OUT
8
13
CS
D
A
OE
BOOT
EPROM
2764
250ns
CLOCK
5
2
RFS
TFS
DT
DR
SCLK
VSS
VDD
CLKOUT
CLKIN
RESET
ADSP-2101
SPORT0
BMS
DATA
ADDRESS
RD
BOOT
EPROM
250 ns

FIG - A·4

APPENDIX – B

SOFTWARE

INTRODUCTION:

This chapter includes all software programs used in this project, namely:

a) Generation of sine wave

b) Graphical Program for monitor display

c) FSK Sample generation program

d) Demodulation of FSK signals (using C language)

e) Demodulation of FSK signals (using ADSP 2101 Assembly language)

f) Floating point number to Hexa decimal number conversion.

g) Datas obtained from the programs.

PROGRAM FOR GENERATING THE SINEWAVE SAMPLES

```
#include<stdio.h>
#include<math.h>
#include<alloc.h>
#include"jo1.c"
float freq,vmax;
float samp_freq;
main()
{

printf("ENTER Vmax\n");
scanf("%f",&vmax);
printf("ENTER FREQ\n");
scanf("%f",&freq);
printf("ENTER SAMPLING FREQ\n");
scanf("%f",&samp_freq);
sine();
jgraph();
}

sine()

{
int j,n;
float k;
double angle;
char file_name[10];
FILE *fp;
printf("ENTER FILE NAME\n");
scanf("%s",file_name);
printf("%s",file_name);

if ((fp=fopen(file_name,"w")) == NULL)
printf("UNABLE TO OPEN\n");
for(n=0;n<64;n++)
{
angle=2*3.14*freq*n/samp_freq;
k=vmax*(double)sin(angle);
fprintf(fp,"%f\n",k);
}
fclose(fp);
}
```

```
  for(i=0;i<1000;i++)
{
  fscanf(fp,"%f",&k);
  setcolor(YELLOW);
  setlinestyle(SOLID_LINE,SOLID_FILL,NORM_WIDTH);
  lineto(2*(i+vp.right/25),k*100.00+vp.bottom/2);
}
  fclose(fp);
  getch();
  closegraph();
}
```

MONITOR DISPLAY PROGRAM FOR DISPLAYING THE GRAPH

```
#include<stdio.h>
#include<stdlib.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
struct viewporttype vp;
jgraph()

{
   int j,i,mode,driver,l;
   char file_name[10];
   float k;
   FILE *fp;
   printf("Type the file name");
   scanf("%s",&file_name);
   fp=fopen(file_name,"r");
   for(i=0,j=0;i<1000;i++,j++)
{
   fscanf(fp,"%f",&k);
   printf("%f\n",k);
   /*if(j=23){j=0,getch();}*/
 }
   getch();
   fclose(fp);
   detectgraph(&driver,&mode);
   initgraph(&driver,&mode,"w:\\software\\tcplus\\bgi ");
   getviewsettings(&vp);
   for(l=0;l<=4;l++)
  { setbkcolor(l);getch();}
   setcolor(GREEN);
   setlinestyle(SOLID_LINE,SOLID_FILL,NORM_WIDTH);
   moveto(0,vp.bottom/2);
   for(i=0;i<=650;i++)
 {
   lineto(i,vp.bottom/2);
 }
   moveto(vp.right/25,50);
   for(i=0;i<=480;i++)
 {
   lineto(vp.right/25,i);
 }
   getch();
   moveto(vp.right/25,vp.bottom/2);
   fp=fopen(file_name,"r");
```

PROGRAM FOR GENERATING FSK SAMPLES

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include "jo1.c"
main()
{
   int x,i,j;
   float k;
   char scan,next;
   FILE *fp,*fp1,*fp2,*fp3,*fp4;
   clrscr();
   fp1=fopen("data","w");
   puts("Enter data ,press 50 when data is over");
   /*scan:scanf("%d",&x);
            if(x==50)
            {fclose(fp1);
            goto next;}
            else
            fprintf(fp1,"%\n",x);
            goto scan;
  next:fp1=fopen("data","r");*/

  while(x!=50)
  {
   scanf("%d",&x);
   if(x==50)
   break;
   fprintf(fp1,"%d\n",x);
  }
  fclose(fp1);
  fp1=fopen("data","r");
  while(!feof(fp1))
  {
  fscanf(fp1,"%d\n",&x);
  printf("%d\t",x);
  }
  fclose(fp1);
  getch();
  fp2=fopen("output","w");
  fp1=fopen("data","r");
  while(!feof(fp1))
```

```
{
fscanf(fp1,"%d\n",&x);
if(x==1)
{
 fp=fopen("s9","r");
 for(i=0;i<64;i++)
 {
+    fscanf(fp,"%f\n",&k);
    fprintf(fp2,"%f\n",k);
  }
  fclose(fp);
 }
 else

 if(x==0)
 {
  fp3=fopen("s7","r");
  for(i=0;i<64;i++)
  {
    fscanf(fp3,"%f\n",&k);
    fprintf(fp2,"%f\n",k);
  }
  fclose(fp3);
 }
  else

  {
   fp4=fopen("rand","r");
   for(i=0;i<=50;i++)
   {
     fscanf(fp4,"%f\n",&k);
     fprintf(fp2,"%f\n",k);
   }
   fclose(fp4);
   }}
   fclose(fp1);
   fclose(fp2);
   jgraph();
   getch();
   }
```

DEMODULATION SOFTWARE (USING 'C' LANGUAGE)

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
static int i,j,l=0,m=1;
FILE *fs,*ft,*fr,*fo;
float x1[700],x2,y1[700],k,x3[700];

fs=fopen("OUTPUT","r");
if(fs==NULL)
{puts("cannot open source file");
goto l1;
}

ft=fopen("S8","r");
if(ft==NULL)
{puts("cannot open source file");
goto l1;
}

fr=fopen("op","w");
if(fr==NULL)
{puts("cannot open target file");
goto l1;
}

fo=fopen("op1","w");
if(fo==NULL)
{puts("cannot open target file");
goto l1;
}

/*reading file1*/
i=1;
while(!feof(fs))
{
fscanf(fs,"%f",&x1[i]);
fflush(stdin);
i+=1;
}

/*reading file2*/
i=1;
```

```
while(!feof(ft))
{
fscanf(ft,"%f",&y1[i]);
fflush(stdin);
i+=1;
}

/*processing & write the value in the file "op" "*/
for(j=1;j<63;j++)
{
x3[j]=(x1[j]+x1[j+1])/2;
k=(x3[j]+y1[j])-x1[j];
k=k*k;
if(k <= 0.04000000)
fprintf(fo,"%d\n",l);
else fprintf(fo,"%d\n",m);
printf("%f\n",k);
fprintf(fr,"%f\n",k);
}
/*closing all files*/
fclose(fs);
fclose(fr);
fclose(ft);
l1: printf("MODIFY THE REQUIRED FILES");
}
```

DEMODULATION SOFTWARE (USING ASSEMBLY LANGUAGE)

```
.MODULE   filter;
.CONST    n=9;
.VAR/DM/RAM  x_input[n];
.INIT     x_input:<hex.dat>;
.PORT     z_out;


      JUMP start;NOP;NOP;NOP;
      RTI;NOP;NOP;NOP;
      RTI;NOP;NOP;NOP;
      RTI;NOP;NOP;NOP;
      RTI;NOP;NOP;NOP;
      RTI;NOP;NOP;NOP;
      RTI;NOP;NOP;NOP;

start: I2=^x_input;
       L2=0;
       M0=1;    L0=0;
       AY1=0;   AY0=0;
       CNTR=n;
       AX0=DM(I2,M0);
       DO div_loop UNTIL CE;
       AR=AX0-AY0;

       AX0=AR;
       AR=AX0+AY1;
       SR=LSHIFT AR BY -1(HI);
       AY1=AX0;

       AX0=DM(I2,M0);
       AX1=AY1;
       AY1=0x01c1;
       DM(z_out)=SR1;
       AR=SR1+AY1;
       AY0=AR;
div_loop:  AY1=AX1;

          IDLE;
     .ENDMOD;
```

SYSTEM FILES FOR EXECUTING THE SIMULATOR

```
bld21 example1
asm21 nco1
ld21 nco1 -a example1 -g -e nco1
sim2101 -a example1
```

```
.SYSTEM example;
.ADSP2101;
.SEG/PM/RAM/ABS=0/CODE/DATA ext_pm[2048];
.SEG/DM/RAM/ABS=14336/data INT_DM[1024];
.PORT/DM/ABS=0      z_out;
.ENDSYS;
```

FLOATING POINT TO HEXA DECIMAL CONVERSION

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
void main()
{
FILE *f1,*f2;
double value,absval,tmp;
float x1[750];
char str[16]; unsigned int X;
int i,j,k;
f1=fopen("output","r");
f2=fopen("hex","w");
i=1;
while(!feof(f1))
{
fscanf(f1,"%f",&x1[i]);
fflush(stdin);
i+=1;
}
for(j=1;j<=i;j++)
{
value=x1[j];
if(value<0) str[0]='1';
if(value>=0) str[0]='0';
absval=fabs(value);
printf("%16g\n",absval);
tmp=absval;
for(k=1;k<=15;k++)
{
tmp=tmp*2;
if(tmp<1) str[k]='0';
if(tmp>=1)
{
str[k]='1';
tmp=tmp-1; } }
printf("%s\n",str);
for(k=0;k<16;k++)
{
X<<=1;
if (str[k]!='0') X|=1;
}
printf("%x",X);
fprintf(f2,"%x\n",X); } }
```

DEMODULATED OUTPUT (BEFORE FIXING THE THRESHOLD)

0.000000	0.007787	0.003247
0.000001	0.169245	0.077314
0.000007	0.190794	0.088261
0.000004	0.019951	0.015960
0.000007	0.068228	0.010052
0.000081	0.225963	0.053862
0.000113	0.117285	0.040998
0.000003	0.001744	0.003327
0.000237	0.157276	0.008699
0.000772	0.213157	0.025540
0.000395	0.035920	0.014981
0.000125	0.048881	0.000773
0.002160	0.221084	0.003185
0.002972	0.141235	0.007807
0.000280	0.000047	0.004689
0.002474	0.134737	0.000617
0.009110	0.223608	0.000186
0.005390	0.054399	0.001140
0.000305	0.031253	0.001399
0.013983	0.209453	0.000972
0.021305	0.163382	0.000352
0.003578	0.003072	0.000001
0.008807	0.111328	0.000348
0.040926	0.229450	0.001126
0.028303	0.075644	0.001262
0.000036	0.016585	0.000462
0.041308	0.191816	0.000011
0.072512	0.183593	0.000752
0.017779	0.011943	0.001537
0.015216	0.081975	0.001088
0.098444	0.223290	0.000139
0.079629	0.101548	0.000217
0.000971	0.003529	0.001196
0.070382	0.150479	0.001501
0.146813	0.189708	0.000642
0.048018	0.031456	0.000000
0.014531	0.037998	0.000608
0.151037	0.174366	0.001488
0.143497	0.118337	0.001225
0.007322	0.001693	0.000242
0.079860	0.073081	0.000120
0.205157	0.144023	0.001056
0.085495	0.052663	0.001541

REFERENCES

1. Modern Analog and Digital Communication Systems, Lathi B.P., Prism Sanders Ed.1993.

2. Principles of Communication Systems. 2nd Ed, Taub, Herbert, and Schilling, Donald.

3. John Prokis Digital Signal Processing.

4. Electronics and Communication in Japan Part-I Vol.69 No.6, 1986.

5. . A survey of Digital phase locked lops Proc. W.C. Lindsey and C.M. Chie, IEEE, April1981.

6. ADSP 2101 Simulator Manual Analog Devices pub.1995.

7. ADSP 21XX users manual Analog Devices Pub.1995.

8. Digital Transmission of Information – Richard E.Blahut Addison Wesley Publish 1990.