INTERFACING AND ANALYSIS OF E1 STREAMS

By

NC Adil Ayub

NC Owais Sami

TC Usman Sharif

TC Furqan Wali

Submitted to Faculty of Computer Science, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of BE Degree in Computer Software Engineering
APRIL 2007

ABSTRACT

INTERFACING AND ANALYIS OF E1 STREAMS

Most telephonic communication is carried out using E1 stream therefore there is a need to analyze this data on the computer system. Computers today, have applications in all kinds of communication systems, and by connecting an E1 stream to a computer, the analysis of a telephonic communication can be done in various ways. This type of interfacing will allow data of an E1 stream to be transmitted into a computer. Different types of analysis can then be done on this data.

E1 stream will be interfaced to the computer through the USB port and the analysis of data and signaling channels will be done by writing software programs.

In order to interface E1 stream to the computer, a circuit will be designed to convert HDB3 bipolar to unipolar and then decode HDB3 line code. Software will be written for frame alignment, recognition of frame, recognition and splitting of channels and graphical representation and analysis of each channel.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of any other award or qualification either at this institution or elsewhere.

DEDICATION

Our project is dedicated to our beloved parents for their prayers and moral support. Without them, we would not have been able to complete our project.

ACKNOWLEDGMENTS

We are eternally grateful to Almighty Allah for bestowing us with the strength and resolve to undertake and complete the project.

We gratefully acknowledge the continuous guidance and motivation provided to us by our project advisors **Lt. Col Mofassir Ul Haque (HOD EE DEPT) and Maj. Taher Ayub (GSO 2 NUST)**. Without their personal supervision, advice and help, timely completion of this project would have been impossible. Our very special thanks are extended to **Head of Department Col. Omar Farooq** for his ever extended moral and technical support. We would also like to thank our esteemed instructors who helped us to nurture our capabilities, giving us the skills and the confidence to take on challenges and emerge victorious.

We are also deeply indebted to our families for their never ending patience and support for our mental peace and to our parents for the strength that they gave us through their prayers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1.1 Introduction

In digital telecommunications, where a single physical wire can be used to carry many simultaneous voice conversations, worldwide standards have been created and deployed. The CEPT originally standardized the **E-carrier** system, which revised and improved the earlier American T-carrier technology, and this has now been adopted by the ITU-T. This is now widely used in almost all countries outside USA, Canada and Japan.

The E-carrier standards form part of the Plesiochronous Digital Hierarchy (PDH) where groups of E1 circuits may be bundled onto higher capacity E3 links between telephone exchanges or countries. This allows a network operator to provide a private end-to-end E1 circuit between customers in different countries that shares single high capacity links in between.

In practice, only E1 (30 circuit) and E3 (480 circuit) versions are used. Physically E1 is transmitted as 32 timeslots and E3 512 timeslots, but one is used for framing and typically one allocated for signaling call setup and tear down. Unlike Internet data services, E-carrier systems permanently allocate capacity for a voice call for its entire duration. This ensures high call quality because the transmission arrives with the same short delay (Latency) and capacity at all times.

E1 circuits are very common in most telephone exchanges and are used to connect to medium and large companies, to remote exchanges and in many cases between exchanges. E3 lines are used between exchanges, operators and/or countries, and have a transmission speed of 34.368 Mbit/s.

1.2 E1 Streams

An E1 link operates over two separate sets of wires, usually coaxial cable. A nominal 2.4 Volt signal is encoded with pulses using a method that avoids long periods without polarity changes. The line data rate is 2.048 Mbit/s (full duplex, i.e. 2.048 Mbit/s downstream and 2.048 Mbit/s upstream) which is split into 32 time slots, each being allocated 8 bits in turn. Thus each time slot sends and receives an 8-bit sample 8000 times per second (8 x 8000 x 32 = 2,048,000). This is ideal for voice telephone calls where the voice is sampled into an 8 bit number at that data rate and reconstructed at the other end.

One timeslot (TS0) is reserved for framing purposes, and alternately transmits a fixed pattern. This allows the receiver to lock onto the start of each frame and match up each channel in turn. The standards allow for a full Cyclic Redundancy Check to be performed across all bits transmitted in each frame, to detect if the circuit is losing bits (information), but this is not always used.

One timeslot (TS16) is often reserved for signaling purposes, to control call setup and teardown according to one of several standard telecommunications protocols. This includes Channel Associated Signaling (CAS) where a set of bits is used to replicate opening and closing the circuit (as if picking up the telephone receiver and pulsing digits on a rotary phone), or using tone signaling which is passed through on the voice circuits themselves. More recent systems used Common Channel Signaling (CCS) such as ISDN or Signaling System 7 (SS7) which send short encoded messages with more information about the call including caller ID, type of transmission required etc. ISDN is often used between the local telephone exchange and business premises, whilst SS7 is almost exclusively used between exchanges and operators. SS7 can handle up to 4096 circuits per signaling channel, thus allowing slightly more efficient use of the overall transmission bandwidth.

Unlike the earlier T-carrier systems developed in North America, all 8 bits of each sample are available for each call. This allows the E1 systems to be used equally well for circuit switches data calls, without risking the loss of any information.

While the original CEPT standard G.703 specifies several options for the physical transmission, almost exclusively HDB3 format is used.

2. Literature Review

2.1 Definitions and Terms.

a. Unipolar Signal. First, we define some terms. If the signal elements all have the same algebraic sign, that is, all positive or negative, then the signal is uni-polar.

b. Polar Signalling. In polar signaling, one logic state is represented by a positive voltage level, and the other by a negative voltage level.

c. Data Rate. The data signaling rate, or just data rate, of a signal is the no of bits per second.

d. Bit Duration. The duration or length of a bit is the amount of time it takes for the transmitter to emit the bit: for a data rate R. the bit duration is l/R.

e. Modulation Rate . The modulation rate, in contrast, is the rate at which signal level is changed: this will depend on the nature of the digital encoding, as explained below. The modulation rate is expressed in bauds i.e. signal elements per second. Finally, the terms mark and space refer to the binary digits 1 and 0, respectively.

f. Register. It is group of binary cells. A cell stores one bit of info. The content of register is of the interpretation given to the info stored in it.

1	0	1	0
1	2	3	4

Figure 2-1 A Four Bit Register.

g. Register Transfer. A basic op in dig system consists of a transfer of info stored in one register into another. The following figure illustrates the transfer of info among registers and demonstrates pictorially the transfer of bin info.

h. Binary Logic. Binary logic deals with variables that take on two discrete values and with operation that assumes logical meaning. The two values the variables take may be called by different names for example, true and false, yes and no. But for our purpose it is convenient to think in terms of bits and assign the values of "1" and "0". Bin logic is used to describe, in a mathematical way, the manipulation and processing of bin info.

i. Logic Gates. Electronic digital circuits are also called logic circuits because with the proper lip, they establish logical manipulation paths. Any desired information for computing or control can be operated upon by passing positive or negative logic as shown in figure 2-2 and 2-3 respectively.

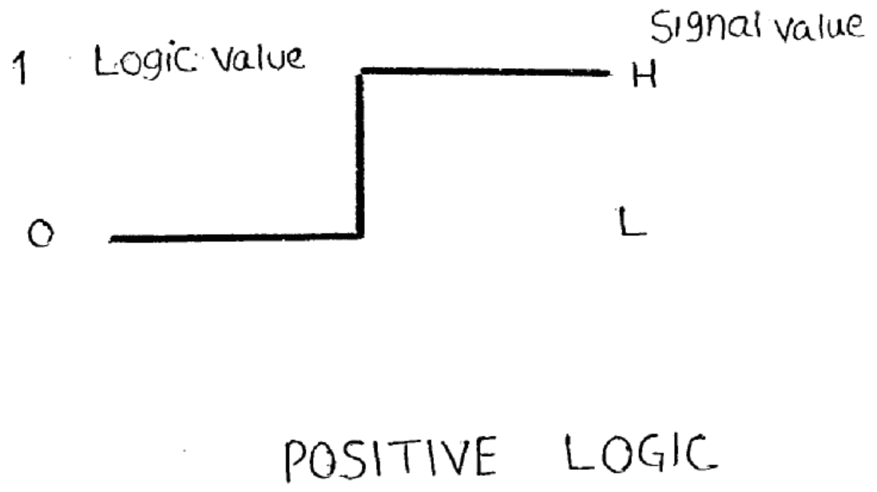


Figure 2-2 Positive Logic

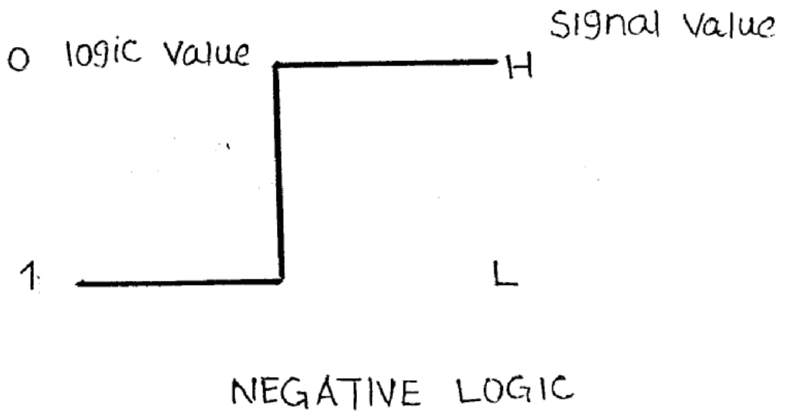


Figure 2-3 Negative Logic

The basic circuit in each family is either a NAND or a NOR gate. The electronic components employed in the construction of the basic circuit are usually used to name the logic family. Different logic families of digital ICs have been introduced commercially. The most popular are as follows:

(1) TTL transistor - transistor Logic

(2) ECL Emitter coupled logic

(3) MOS Metal oxide semi conductor.

(4) CMOS Complementary Metal oxide semi- conductors.

j. **TTL.** Numerical designation as the 5400 and 7400 series usually distinguish these ICs. The 5400 series has a wide operating temp range suitable for industrial use. The later has a narrower temp range, suitable for industrial use. The numeric designation of 7400 series means that IC packages are numbered as 7400 and 7402 etc. Some vendors make available TTL ICs, with different numeric designations such as 9000 or 8000 series.

k. **Fan Out.** It specifies the no of STD loads that the o/p of a gate can drive without impairing its normal op. A STD load is usually defined as the amount of current needed by an i/p of an other gate in the same IC family. Sometimes the term loading is used instead of fan-out. This term is derived from the fact that the o/p of a gate can supply a limited amount of current above which it seizes to operate properly and is said to be over loaded.

if Q=1 it switches to Q=O and vice versa. A clocked JK Flip flop logic diagram is shown in figure 2-4, graphic symbol is shown in Table 2-1.

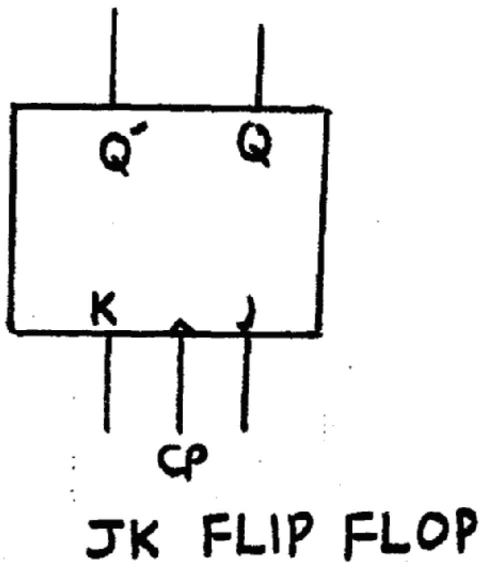
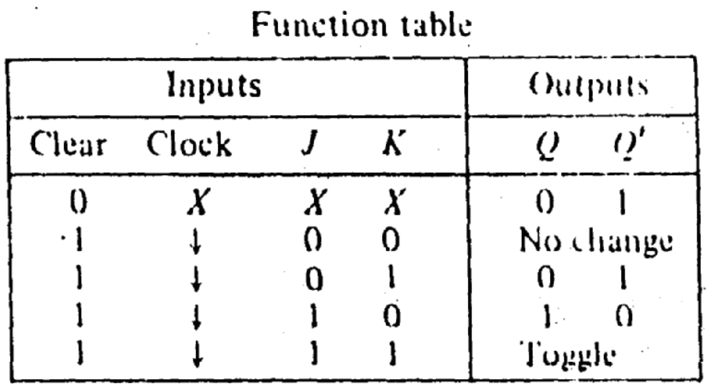


Figure 2-4 JK Flip Flop


Function table

Inputs				Outputs	
Clear	Clock	J	K	Q	Q'
0	X	X	X	0	1
1	↓	0	0	No change	
1	↓	0	1	0	1
1	↓	1	0	1	0
1	↓	1	1	Toggle	

***JK* flip-flop with direct clear**

Table 2-1 JK Flip-Flop with Direct Clear

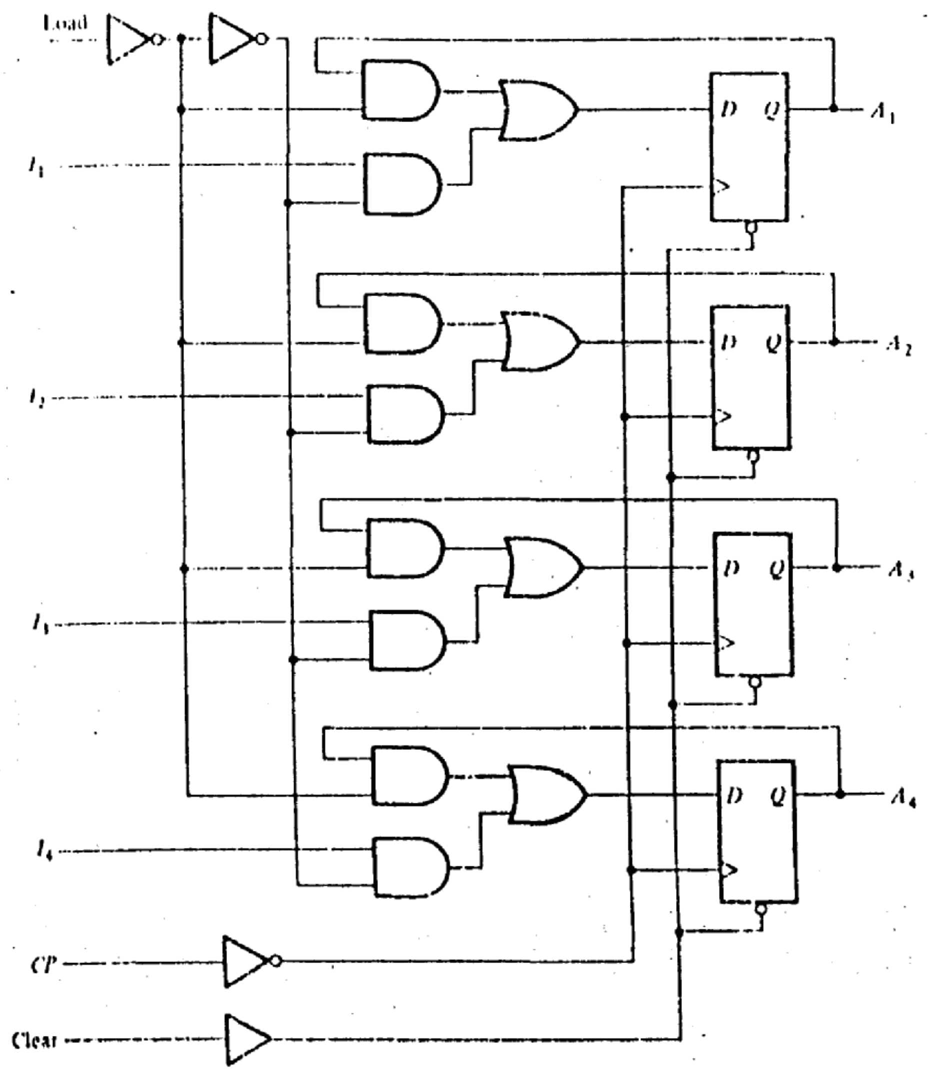
l. T Flip Flop. The T flip flop is the Single I/P version of 1K flip flop. The designation T comes from the ability of flip flop to toggle or change state. Regard less of the present state of the flip flop, it assumes the complement state when the clock pulse occurs while I/P T is at logic 1.

m. **Triggering Of Flip-Flops.** The state of a flip-flop is switched by a momentary change in the input signal. This momentary change is called a trigger and the transition it causes is said to trigger the flip-flop. Asynchronous flip-flops, require an input trigger defined by a change of signal level. This level must be returned to its initial value (0 in the NOR and 1 in the NAND flip-flop) before a second trigger is applied. Clocked flip-flops are triggered by pulses. A pulse starts from an initial value of 0, goes momentarily to 1, and after a short time, returns to its initial 0 value. The time interval from the application of the pulse until the output transition occurs is a critical factor that needs further investigation. A clock pulse may be either positive or negative. A positive clock source remains at 0 during the interval between pulses and goes to 1 during the occurrence of a pulse. The pulse goes through two signal transitions, from 0 to 1 and then returns from 1 to 0. Positive transition is defined as the positive edge and the negative transition as the negative edge.

N. **Edge-Triggered Flip-Flop.** Another type of flip-flop that synchronizes the state changes during a clock pulse transition is the edge-triggered flip-flop. In this type of flip-flop, output transitions occur at a specific level of the clock pulse. When the pulse input level exceeds this threshold level, the inputs are locked out and the flip-flop is therefore unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs. Some edge-triggered flip-flops cause a transition on the positive edge of the pulse, and others cause a transition on the negative edge of the pulse.

o. **Direct Inputs.** Flip-flops available in IC packages sometimes provide special inputs for setting or clearing the flip-flop asynchronously. These inputs are usually called direct preset and direct clear. They affect the flip-flop on a positive (or negative) value of the input signal without the need for a clock pulse. These inputs are useful for bringing all flip-flops to an initial state prior to their clocked operation.

p. **Register With Parallel Load.** The transfer of new information into a register is referred to as loading the register. If all the bits of the register are loaded simultaneously with a single clock pulse, we say that the loading is done in parallel. A pulse applied to the CP input of the register will load all four inputs in parallel. This is shown in figure 2-5.



Register with parallel load using *D* flip-flops

Figure 2-5 Register with Parallel Load using D Flip Flops

q. Shift Registers. A register capable of shifting its binary information either to the right or to the left is called a shift register. The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive a common clock pulse, which causes the shift from one stage to the next. This is shown in figure 2-6

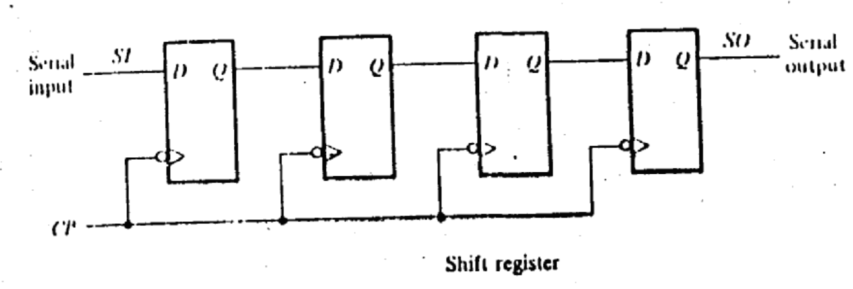


Figure 2-6 Shift Registers

2.2 Serial Transfer

A digital system is said to operate in a serial mode when information is transferred and manipulated one bit at a time. The content of one register is transferred to another by shifting the bits from one register to the other The information is transferred one bit at a time by shifting the bits out of the source register into the destination register.

2.3 Binary Ripple Counter

A binary ripple counter consists of a series connection of complementing flip-flops (I or JK type), with the output of each flip-flop connected to the CP input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming count pulses. The diagram of a 4- bit binary ripple counter is shown in Fig 2-7. All J and K inputs are equal to 1. The small circle in the CP input indicates that the flip-flop complements during a negative-going transition or when the output to which it is connected goes from 1 to 0. The flip-flop change one at a time in rapid succession, and the signal propagates through the counter in a ripple fashion. Ripple counters are sometimes called asynchronous counters.

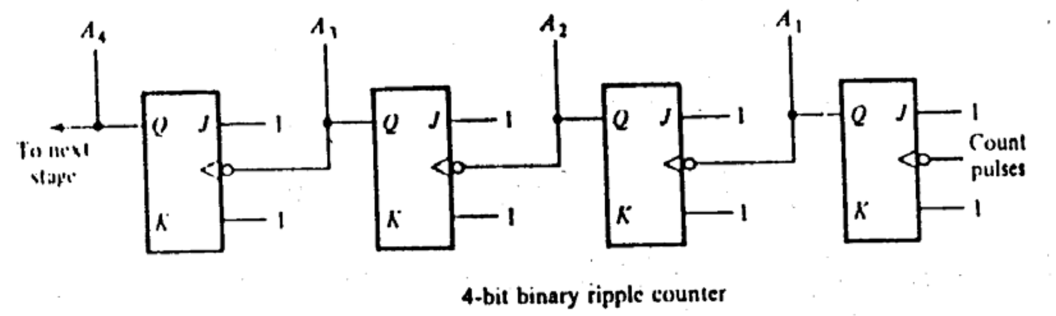


Figure 2-7 4-Bit Binary Ripple Counter

2.4 Interpretation of Received Digital Signal

Referring to Figure 2-7 can summarize the tasks involved **in** interpreting digital signals at the receiver.

(1) The receiver must know the timing of each bit. That is, the receiver must know with some accuracy when a bit begins and ends.

(2) The receiver must determine whether the signal level for each bit position is high (1) or low (0). In Figure S sampling each bit position in the middle of the interval and comparing the value to a threshold performs these tasks. Because of noise and other impairments, there will be errors, as shown.

Factors to determine the successful interpretation of the incoming signal:

(1) The signal-to-noise ratio

(2) The data rate

(3) The bandwidth with other factors held constant, the following statements are true

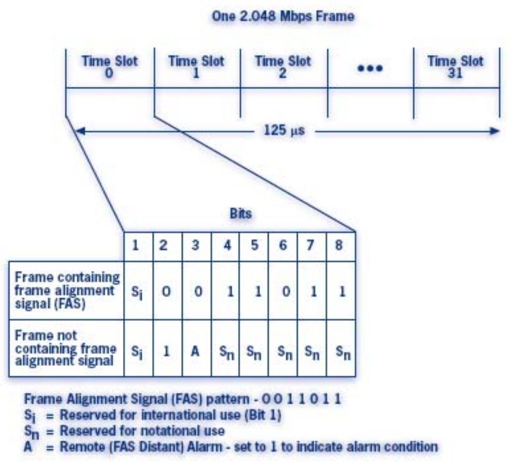
2.5 Cost and Complexity

Although digital logic continues to drop in price, expense should not be ignored. In particular, the higher the signaling rate to achieve a given data rate, the greater the cost.

3. Framing

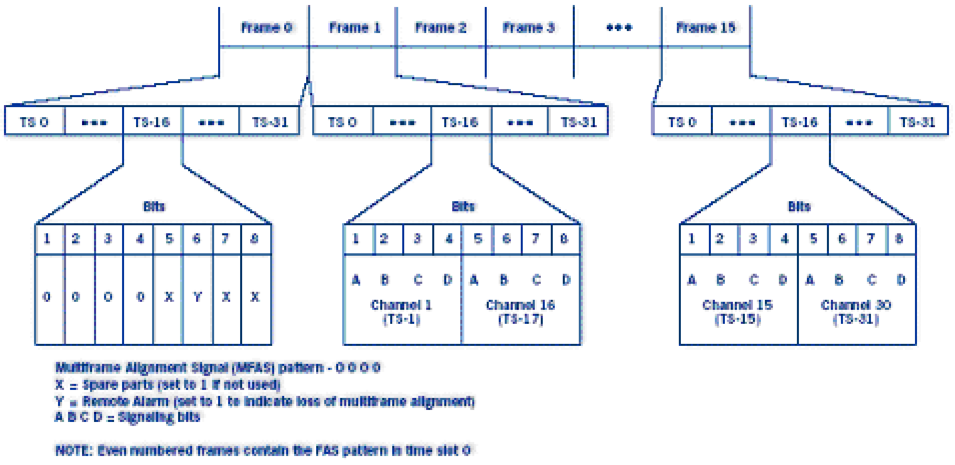
3.1 The 2.048 Mbps Framing Format

The 2.048 Mbps signal typically consists of multiplexed data and/or voice which requires a framing structure for receiving equipment to properly associate the appropriate bits in the incoming signal with their corresponding channels. Figure 3-1 shows the framing for the 2.048 Mbps signal as defined in ITU-T Recommendation G.704. As can be seen in Figure 3-1, the 2.048 Mbps frame is broken up into 32 timeslots numbered 0-31. Each timeslot contains 8 bits in a frame, and since there are 8000 frames per second, each time slot corresponds to a bandwidth of 8 x 8000 = 64 kbps. Time slot 0 is allocated entirely to the frame alignment signal (FAS) pattern, a remote alarm (FAS Distant Alarm) indication bit, and other spare bits for international and national use. The FAS pattern (0011011) takes up 7 bits (bits 2-8) in timeslot 0 of every other frame. In those frames not containing the FAS pattern, bit 3 is reserved for remote alarm indication (FAS Distant Alarm) which indicates loss of frame alignment when it is set to 1. The remaining bits in timeslot 0 are allocated as shown in Figure 3-2. If the 2.048 Mbps signal carries no voice channels, there is no need to allocate additional bandwidth to accommodate signaling. Hence, time slot 1-31 are available to transmit data with an aggregate bandwidth of 2.048 Mbps - 64 kbps (TSO) = 1.984 Mbps.


One 2.048 Mbps Frame
Time Slot 0
Time Slot 1
Time Slot 2
•••
Time Slot 31
125 µs
Bits
1 2 3 4 5 6 7 8
Frame containing frame alignment signal (FAS)
Si 0 0 1 1 0 1 1
Frame not containing frame alignment signal
Si 1 A Sn Sn Sn Sn Sn
Frame Alignment Signal (FAS) pattern - 0 0 1 1 0 1 1
Si = Reserved for international use (Bit 1)
Sn = Reserved for notational use
A = Remote (FAS Distant) Alarm - set to 1 to indicate alarm condition

The 2.048 Mbps framing format

Figure 3-1 2.048MBPS Framing Format



Multiframe Alignment Signal (MFAS) pattern - 0 0 0 0
X = Spare parts (set to 1 if not used)
Y = Remote Alarm (set to 1 to indicate loss of multiframe alignment)
A B C D = Signalling bits

NOTE: Even numbered frames contain the FAS pattern in time slot 0

The 2.048 Mbps TS-16 multiframe format

Figure 3-2 2.048 Mbps TS-16 Multi Frame Format

If there are voice channels on the 2.048 Mbps signal, it is necessary to take up additional bandwidth to transmit the signalling information. ITU-T Recommendation G.704 allocates time slot 16 for the transmission of the channel-associated signalling information. This is explained in the next section.

3.2 The 2.048 Mbps TS-16 Multiframe Format

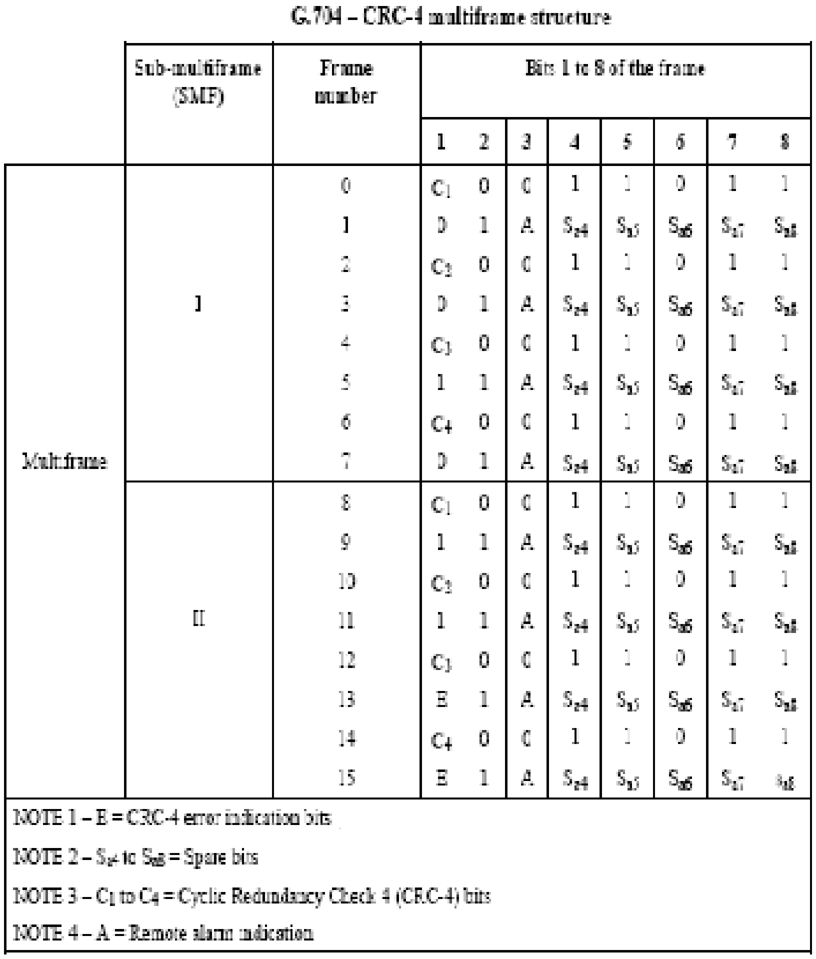
The 2.048 Mbps can carry up to thirty 64 kbps voice channels in time slot 1-15 and 17-31. Voice channels are numbered 1-30; voice channels 16-30 are carried in time slot 17-31.

However, the 8 bits in time slot 16 are not sufficient for all 30 channels to signal in one frame. Therefore, a multiframe structure is required where channels can take turns using time slot 16. Since two channels can send their ABCD signalling bits in each frame, a total of 15 frames are required to cycle through all of the 30 voice channels. One additional frame is required to transmit the multiframe alignment signal (MFAS) pattern, which allows receiving equipment to align the appropriate ABCD signalling bits with their corresponding voice channels. This results in the TS-16 multiframe structure where each multiframe contains a total of 16 2.048 Mbps, numbered 0-15. ***Figure 3-2*** on the previous page shows the TS-16 multiframe format for the 2.048 Mbps signal as defined by the ITU-T Recommendation G.704. As can be seen in ***Figure 3-2***, time slot 16 of frame 0 contains the 4-bit long multiframe alignment signal (MFAS) pattern (0000) in bits 1-4. The "Y" bit is reserved for the remote alarm (MFAS Distant Alarm) which indicates loss of multiframe alignment when it is set to 1.

Time slot 16 of frames 1-15 contains the ABCD signalling bits of the voice channels.
Time slot 16 of the nth frame carries the signalling bits of the nth and (n+15)th voice channels. For example, frame 1 carries the signalling bits of voice channels 1 and 16, frame 2 carries the signalling bits of channels 2 and 17 etc. It is also important to note that the frame alignment signal (FAS) is transmitted in time slot 0 of the even numbered frames. We have thus explained how frame alignment and channel associated signaling are achieved in 2.048 Mbps transmission. (Alternatively, time slot 16 may also be used for common channel signalling applications such as primary rate ISDN). It must be noted, however, that the 2.048 Mbps framing and TS-16 multiframing structures discussed so far do not provide any built in error detection capabilities, which could be used to determine the error performance of the 2.048 Mbps system on an in-service basis. This capability is provided by the CRC (Cyclic Redundancy Check) multiframe structure as explained in the next section.

3.3 The 2.048 Mbps CRC Multiframe Format

This section describes the specifics of the 2.048 Mbps CRC Multi frame format. The 2.048 Mbps CRC Multi frame structure as defined by ITU-T Recommendation G.704 is shown in ***Table3-1*** on the previous page. The CRC Multi frame consists of 16 frames (numbered 0-15) which are divided into two sub-multi frames (SMF-1 and SMF-11) of 8 frames each. The 4-bit long CRC word associated with each sub multi frame, SMF (N) is inserted into the next sub-multi frame, SMF (N+1). The CRC bits take up the 1st bit of time slot 0s containing the 7-bit FAS (Frame Alignment Signal) pattern. The CRC Multi frame alignment signal uses the 1st bit of time slot 0s not containing the FAS pattern.(See ***Table 3-1***).



G.704 – CRC-4 multiframe structure

	Sub-multiframe (SMF)	Frame number	Bits 1 to 8 of the frame							
			1	2	3	4	5	6	7	8
Multiframe	I	0	C_1	0	0	1	1	0	1	1
		1	0	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
		2	C_2	0	0	1	1	0	1	1
		3	0	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
		4	C_3	0	0	1	1	0	1	1
		5	1	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
		6	C_4	0	0	1	1	0	1	1
		7	0	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
	II	8	C_1	0	0	1	1	0	1	1
		9	1	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
		10	C_2	0	0	1	1	0	1	1
		11	1	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
		12	C_3	0	0	1	1	0	1	1
		13	E	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}
		14	C_4	0	0	1	1	0	1	1
		15	E	1	A	S_{a4}	S_{a5}	S_{a6}	S_{a7}	S_{a8}

NOTE 1 – E = CRC-4 error indication bits

NOTE 2 – S_{a4} to S_{a8} = Spare bits

NOTE 3 – C_1 to C_4 = Cyclic Redundancy Check 4 (CRC-4) bits

NOTE 4 – A = Remote alarm indication

Table 3-1 CRC4 Multi Frame Structure

3.4 Cyclic Redundancy Check (CRC)

3.4.1 Multiplication/Division Process

A particular CRC-4 word, located in sub-multi frame N, is the remainder after multiplication by x4 and then division (modulo 2) by the generator polynomial x4̣ x̣ 1, of the polynomial representation of sub-multi frame N – 1.

When representing the contents of the check block as a polynomial, the first bit in the block, i.e. frame 0, bit 1 or frame 8, bit 1, should be taken as being the most significant

bit. Similarly, C1 is defined to be the most significant bit of the remainder and C4 the least significant bit of the remainder.
There may be a need to update CRC-4 bits at intermediate equipments which access the Sa4 bit message-based data-link (see 3.4.2).

3.4.2 Encoding Procedure

The CRC-4 bits in the SMF are replaced by binary 0s. The SMF is then acted upon by the multiplication/division process. The remainder resulting from the multiplication/division process is stored, ready for insertion into the respective CRC-4 locations of the next SMF. The CRC-4 bits thus generated do not affect the result of the multiplication/division process in the next SMF because, as indicated above, the CRC-4 bit positions in an SMF are initially set to 0 during the multiplication/division process.

3.4.3 Decoding Procedure

A received SMF is acted upon by the multiplication/division process, after having its CRC-4 bits extracted and replaced by 0s.The remainder resulting from this division process is then stored and subsequently compared on a bit-by-bit basis with the CRC bits received in the next SMF. If the remainder calculated in the decoder exactly corresponds to the CRC-4 bits received in the next SMF, it is assumed that the checked SMF is error free.

3.5 Signaling Bits (a b c d) in TS16

There are a number of protocols which can run on top of E1.These protocols are grouped into 3 big subgroups, CAS, CCS, and RBS.

3.5.1 CAS Signaling

CAS stands for Channel Associated Signaling. Examples are FXS loop start and E&M wink start. These protocols provide information such as the number that was called, and what state the call is in. With this kind of signaling, a set of bits is used to replicate opening and closing the circuit (as if picking up the telephone receiver and pulsing digits on a rotary phone), or using tone signaling which is passed through on the voice circuits themselves.

CAS is the original signaling system used by E1. In CAS, Timeslot16 is reserved for signaling.

For digital E1 Channel Associated Signaling (CAS) trunks that run ear and mouth (E&M) signaling, there are generally only two states in which a voice channel can be. When there is no call on a channel, the channel is in the Idle, or On- Hook state. When there is an active call on a channel, then the channel is in the Seized, or Off-Hook state. Table 3-2 shows the standard Transmit/Receive ABCD signaling bit patterns for the Idle and Seized states:

Direction	State	A	B	C	D
Transmit	Idle/On-Hook	0	0	0	0
Transmit	Seized/Off-Hook	1	1	1	1
Receive	Idle/On-Hook	0	0	0	0
Receive	Seized/Off-Hook	1	1	1	1

Table 3-2 Transmit/Receive ABCD Signaling Bit Patterns

After a channel is initially seized, each device must indicate the progress of a call. The progress indicators include whether a call is answered or remains unanswered, and when a call is answered, which party disconnects first. These call progress states are important as Telephony systems need to know when the call was attempted, answered, and cleared.

4. Hardware

4.1 Scheme of Progression

Interfacing of E1 stream with the computer requires the following steps :

1. Conversion of HDB3 Bipolar to HDB3 Unipolar.
2. Generation of clock signal
3. HDB3 unipolar RZ to HDB3 unipolar NRZ
4. HDB3 unipolar NRZ - Decoding.
5. Conversion of +HDB3 and -HDB3 NRZ signal to data signal.
6. Interfacing micro controller to computer

4.2 Block Description

4.2.1 Description

Step 1. Conversion of HDB-3 Bipolar to HDB-3 unipolar. HDB-3 Bipolar signals have and -“pulses. These have level of 3 “V for negative pulses and positive “3”V for positive pulses. TTL logic level does not accept -ive voltages. Therefore we segregate negative and positive pulses of HDB-3 signals. Both are represented as ‘0’ and ‘1’. “0” level is at zero volt that is low and “l” is at 5 volts that is high.

Step 2. Generation of clock signal using an IC 17555, we can obtain 2.048 MHz clock by altering the two resistance values that is 1 kilo ohm and 470 ohm.

Step 3. HDB-3 unipolar RZ to HDB-3 unipolar NRZ. In RZ unipolar Signal one bit is rep resented by half clock cycle and in NRZ, one bit is represented by one full clock cycle. So we converted RZ pulses into NRZ.

Step 4 HDB-3 unipolar NRZ - Decoding Violation format is deleted from HDB-3 +ve and HDB-3 (-ive). The violation format is 000V, BOOV.

Step 5 Conversion of NRZ to Data Signal +ve and -ive pulses of decoded
HDB-3 are 'OR'ed to get the actual data bits.

Step-6 The 'OR'ed signal from step 5 is input to the micro-controller at RAO input. The firmware code (Appendix A) is burnt in the controller making cable is attached with RC5 (D+), RC4 (D-), Vcc and ground.

4.2.2 Impedance Matching

Impedance matching is achieved by using a transformer circuit. Impedance of 130 ohms is matched with i/p impedance of RL 421 interface circuit. By using the same transformer, we converted the unbalanced signal to balance signal.

4.3 Detailed Description of Circuit

4.3.1 HDB-3 Receiver Circuit.

The HDB-3 signal received from TLU 20/TLU 64 of PASCOMS **(LIM)** is a balanced signal. It is required to be converted into unbalanced form. The difference in balanced and unbalanced signal is attached as per Anx A. It is converted by using transformer. The transformer isolates i/p and o/p of the interface at the same time, it matches the impedance of 75 ohms. Which is achieved T by. using a resistor R1 across the secondary winding of transformer. Terminal T-8 of transformer TI is connected to ground. The 2uid terminal 1-5 is connected to the pin number 2 and pin number 7 of two op amp in IC 26C32. These two op amps are being used as comparator. Voltage protection circuit is also provided which protects the over voltage, more than 3V.

4.3.2 HDB-3 Signal (Bi-Polar) to HDB 3 (+) and HDB-3 (-) Unipolar Signals

Two comparators are used comparator A is for +ve HDB-3 Signal and comparator **B** is for -ive HDB-3 Signal. A voltage divider circuit co1sisting of R-2 and R-3 gives constant 2 volts at pin no27of IC 1 of both comparators A&B. These two volts are used when *i/p* is at zero volt so that both comparators should give “0” volt at their o/p. These two comparators are used, their o/p to be compatible with TTL logic level. The 2nd voltage divider NW consists of R5, R6 and R7. It provides 2.7 volts and 1.8 volt at the inverting i/p of comparator A and non inverting i/p of comparator B, respectively. 2.7 volts are taken across R6+R7, and 1.8 volts are taken across R7.When *i/p* voltage of +3 volt, are applied at the non inventing i/p of comparator A and inverting i/p of comparator B, these two voltages are compared with fix voltages already applied to comparators. Therefore the comparator A’s o/p becomes high and o/p of comparator B goes low, Because the voltages at inverting terminal of comparator B are higher than the fix voltages at non inventing terminal of comparator B. Similarly, when i/p of -ive voltage, that is —3V is applied at the comparators, these are compared with fix voltages already applied to both comparators. The o/p of comparator B becomes high and o/p of comparator A goes low. In this way, the negative pulse of HDB-3 signal is converted to +ve pulses

4.3.3 HDB-3 Decoder.

HDB-3 (+ve) and (-ive) in RZ form are applied to D flip flops Dl and D2 respectively. These two flip flops convert the HDB-3 (+ve) and HDB-3 (-ive) RZ signal to NRZ form. It is then applied to the 4 bit serial shift register. In these two flip flops, the data signal is aligned with the clock pulses of 2.048 MB/Sec. The clock signal is coming from clock generator module. The 4 bit shift register consisting of IC 74179 (IC4) is (-ive) edge triggered. When all the four registers are loaded, logic circuit connected to o/ps of all registers decides whether the loaded signal should be decoded to all “Os” or not. The logic circuit consists of 1C5 (7421) and 1C7 (7432) and 1C3 (7404). This logic circuit senses two sequences of 1001 or 0001. It then coverts them to all the 0s. When all the registers are loaded with above sequence the o/p of logic circuit goes high. That clears all the registers on the next coming clock pulse. The o/p of shift register taken from pin number 11 is applied to the i/p pin number 12 and 2 of Dl and D2 flip flops (IC6). These two flip fops are +ve edge triggered. The o/p of Dl flip flop is +ve HDB-3 decoded NRZ and o/p of D2 is -ive HDB 3 decoded NRZ signal. The o/p of Dl and D2 at pin number 9 and 5 of 1C6 (7474) are applied at i/p terminals 9 and 10 of OR gate of 1C7 (7432). The o/p of OR gate is NRZ data. See Appendix B.

5. Software

5.1 Introduction

The software portion consists of two main classes in Matlab.

Namely:

1. Gui.m
2. Process.m

5.1.1 Gui.m consists of the GUI (Graphical User Interface). It integrates a window that allows us to analyze our data coming as the E1 data. It calls several functions to get this data, plot the graphs, and open various channels.

5.1.2 Process.m is the class that acts as a client. A server is generating the e1 data, and the client initiates a connection with the server. It retrieves the data from the server, and after frame alignment and channel splitting, displays it in the channel windows. –

In the forthcoming section, we will describe the software phase in complete detail.

5.2 Prerequisites

5.2.1 E1 data that we are generating is random. E1 binary stream has a permanent inherent pattern. Using this, we have developed an algorithm that generates a stream of binary values that have the characteristics of an e1 binary stream. Our Server is the program actually creating this data randomly.

To add to this, we have also written algorithm to deal with a real time data. If a stream is actually coming, and we have it stored it in a file. We have also developed an algorithm that will take the data values coming from a real e1 binary stream, and will be able to act as a server and send them to the client.

5.2.2 On retrieval of E1 data there are a number of steps that are needed to be done to start working with it. As our hardware portion does the decoding of the e1 stream. The next few steps are done by our software phase.

5.3 Modules

Code has been classified into a few modules:

- **5.3.1 Server**
    - Generating E1 data (VC/C++ 6 application)
    - (Optional) E1 Retriever.
- **5.3.2 Client**
    - Graphical User Interface (gui.m)
    - Acquiring E1 data (process.m).
    - Frame Alignment (process.m).
    - Multi Frame Alignment (process.m).
    - Frame Recognition and Splitting (tfallign.m).
    - Channel Recognition and Splitting (chspliter.m).
    - Retrieval of CRC bits from TS0 of every frame in a super frame (crc_finder.m).
    - Retrieval of Signaling bits (a b c d) from TS16 of every frame in a super frame (siginfo.m).
    - Graphical Display of data (plottd.m, plotfd.m, crcinfogui.m, siginfogui.m, ftransform.m)

5.4 Server Modules

5.4.1 E1 Generator (Visual C/C++ 6)

To start working on software portion we generated random E1 data on Local Host i.e. 127.0.0.1 and port 231. This application is named as E1 Generator and is made in VC++6.

- It first makes a server on Internet protocol (IP) 127.0.0.1 and port 231 and waits for the client (Matlab) to send a request for data.
- When the client writes '1' on this server the server gets prepared to send the data.
- **GetE1_data** generates random e1 data in between 0 to 255 values and puts it in a circular buffer.
- **ReadE1_Data** then reads values from the circular buffer and also appends 0x9B (155 decimal, 10011011 binary) or 0x1B (27 decimal, 00011011 binary) i.e. the frame alignment signal (FAS) after every 512 bits or 64 bytes.
- It also appends the multi frame alignment signal 0000 binary in every TS16 of frame 0 in super frame (MFAS).
- This is how the data has e1 characteristics as after every 512 bytes it has a frame alignment signal (FAS).
- After reading the data, **ReadE1_Data** function sends this data to the client.

The Algorithm for this process is given in Table 5-1 given in Appendix C.

5.4.2 E1 Retriever (Visual C/C++ 6)

Once decoded E1 binary stream of data, is stored in a text file(source file), the program can be used to retrieve it, and send it for analysis. In this portion the program acts as a server to Matlab (client) on Local Host i.e. 127.0.0.1 and port 231. This application is named as E1 Retriever and is made in VC++6. It includes the following steps:

- It first makes a server on Internet protocol (IP) 127.0.0.1 and port 231 and waits for the client (Matlab) to send a request for data.
- When the client writes ‘1’ on this server the server gets prepared to send the data.
- **GetE1_data** reads 8 bits from the source file and stores it in a circular buffer.
- **ReadE1_Data** gets the data from the buffer and sends to client (Matlab).

The algorithm is shown in Table 5-2 given in Appendix C.

5.5 Client Modules in Detail

5.5.1 Gui.m

The Graphical User Interface (GUI) executed through the gui.m function after clicking the Acquire Data button Process.m function runs. In this function all the processing is done, it can be considered the basic backbone of the entire system.

The other three buttons in the GUI are

1. Time domain button
2. Frequency domain button
3. Stop button

The **Time domain** button calls the plottd.m function which draws all the graphs, in time domain i.e. time vs. amplitude and displays them on the main GUI window.

Similarly the **Frequency domain** button calls plotfd.m function which draws the frequency domain graphs i.e. frequency vs. power, in the form of stem plotting, and displays them on the main GUI window.

Stop button asks the user to stop/close the entire analysis portion.

The Menu Bar consists of

- Channel Info
- CRC bits Info
- Signaling Info
- Help
- Exit

Channel Info prompts a dialog box, which requires the user to input a certain channel number between 1 & 32. There are two conditions to be checked before giving information for a certain channel. First it needs to make sure that frame alignment has been done, and there is sufficient data to analyze a single channel. Frame alignment requires that at least one super frame of E1 binary stream has been received. If channel info is required before frame alignment, an error dialog will be displayed.

Secondly, once sufficient data is present, it checks if the input is outside the range 1-32. If yes, an error dialog box is displayed. Otherwise if a correct value is entered within the range, it calls **ftransform.m** which displays all type of graphical analysis for that particular channel number which was entered (ftransform.m will be explained in the forthcoming section).

CRC bits Info can only be displayed if at least once super frame has been received. Once a super frame has been received, it means that frame alignment has been done, and 128 bits of values are stored for channel 1 (16x8 bits of TS0). CRC bits will be extracted through **crc_finder.m** and would be displayed for each super frame. For further information refer to section 3.3.

Signaling Info can only be displayed if at least once super frame has been received. Once a super frame has been received, it means that frame alignment has been done, and 128 bits of values are stored for channel 17 (16x8 bits of TS16). Signaling bits will be extracted through **siginfo.m** and would be displayed for each super frame. For further information refer to section 3.5.

Help will explain the functionality of the Graphical User Interface (GUI).

Exit asks the user to stop/close the entire analysis portion.

5.5.2 Process.m

First we initialize the TCP/IP (Transmission Control Protocol/Internet Protocol) interface through the **TCP_Init.m** function. In the TCP function it makes an object named as e1_serv, and binds it to the local host i.e. 127.0.0.1 and port 231, and opens the port. Now the program is acting as a client.

Then this client sends "1" on the server through **SendBeginDataTX_Req.m** function, so the server can initialize its transmission of e1 binary stream. Next, an infinite loop starts reading this data. The function used in this loop is **ReadE1_data.m** function. The **ReadE1_data.m** function will receive 32 bits at one time. The received data will be in decimal format (i.e. 4x8=32 bits). Since the server is sending data in decimal format, we convert it to binary format using function **tobinary.m.**

The function **tobinary.m** is performing the following tasks:

- It converts the decimal data to binary.
- If the data is not comprised of 8 bits, 0's are appended in the Least Significant Bits (LSB), to complete 8 bits.
- These string values are then converted to numeric values for processing.

The retrieved binary data is then displayed in the main GUI window. All this data is continuously being stored in a source file named "source.txt". During execution, variable named 'x' is incremented (by 1) each time the data is read. When x reached a value of 128, 4096 bits have been received i.e. one super frame.

When x=1	4	decimal values are read
When x=128	4*128 =512	decimal values are read
		(Each decimal value comprises 8 bits. Decimal values sent by generator range is 0-255)

i.e. 512*8 (bits) = 4096 bits (or one super frame)

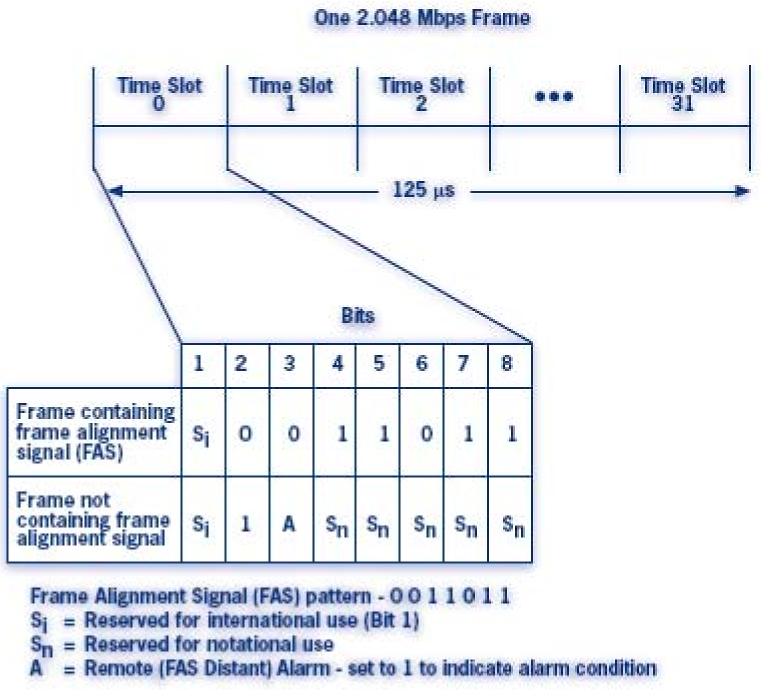
When x=128 frame alignment function **tfallign.m** runs. The algorithm is explained in Table 5-3. (**tfallign.m** will be explained in the forthcoming section).

```
if(first ==1 & x >= 128 & y>=128)


       if(do_allign==1)
          position=tfallign(prev_allig,handles);
          if(position ~=0)
            do_allign=0;
            remm=rem(len_of_B,256);

            if(remm ~= 0)
              ss=256-remm;
              pos_in_source=pos_in_source+ss;

              if(ss>32)
                yy=ss/32;
                yy=ceil(yy);
                x=x-yy;
              else
                x=x-1;
              end

            end
          end

       end
```
Table 5-3 Receiving Super Frame.

The **tfallign.m** function returns the starting position of the TS0 of frame 0 in the super frame. This returned position is given to another function named **getbits1.m** as its argument. The **getbits1.m** function takes this argument as a starting position in **source.txt**, and the length of this file as the ending position and copies all the data between them to a new file called **mainsource.txt.**

The file **mainsource.txt** contains the valid e1 data that is to be split into channels, and thus analyzed. If the **mainsource.txt** does not contain an amount of data that could be equally distributed among 32 channels then a flag is set as true, and the required amount of bits are extracted from the server and stored in **source.txt.** Then the function **getbits2.m** is called. This function is used to extract the exact amount of bits from **source.txt** which will complete **mainsource.txt** to have values exactly divisible to 32 channels. Note that this function **getbits2** will only execute in case where **mainsource.txt** has such amount of data that could not be split equally among the 32 channels.

The **tfallign.m** function returns the starting position of the TS0 of frame 0 in the super frame. In every odd frame of the super frame there is a yellow alarm signal. Since this is an odd frame, so the position returned by **tfallign.m** +258 is the yellow alarm bit. This is shown in the figure 5-1 below.



	1	2	3	4	5	6	7	8
Frame containing frame alignment signal (FAS)	S_i	0	0	1	1	0	1	1
Frame not containing frame alignment signal	S_i	1	A	S_n	S_n	S_n	S_n	S_n

Frame Alignment Signal (FAS) pattern - 0 0 1 1 0 1 1
S_i = Reserved for international use (Bit 1)
S_n = Reserved for notational use
A = Remote (FAS Distant) Alarm - set to 1 to indicate alarm condition

The 2.048 Mbps framing format

Figure 5-1. Yellow Alarm

Next alarm signal will come after 512 bits. For all the super frames (4096 bits Frame 0 - Frame 15) collected so far in the source.txt, this yellow alarm condition is checked. The last alarm signal will be present in the TS0 of frame 15(last odd frame). If this alarm bit is 1, then the yellow alarm indicator in the GUI window is turned on. Otherwise if it is 0, there is no yellow alarm present.

Now the channel splitting algorithm runs, and the position returned by **tfallign.m** is given as the argument to this function named **chspliter.m.** This function returns the last position till which the data is split.

Next, each time 256 bits of data is received and stored in **source.txt**, it is checked whether these 256 bits are of even or odd frame. If the frame is odd, it will contain the yellow alarm bit. Alarm bit is checked, and a 1 will indicate a yellow alarm in GUI, while a 0 will display nothing. Otherwise if it is an even frame, the frame alignment signal is checked once again in the **alignment_testeven.m** function. This time if these 256 bits still have the correct frame alignment, these 256 bits are taken from **source.txt** and stored into **mainsource.txt.** Then the function **chspliter.m** splits these 256 bits into 32 separate channels of 8 bits each. Otherwise if the condition is false, then we collect 4096 bits of data i.e. receive a super frame from the server and store them in **source.txt**. The alignment function **tfallign.m** will run again with take the last position of the previously aligned data as input and will return a new aligned position. If it fails to find an aligned position then it will again collect one more super frame i.e. 4096 bits from the server and this will continue until it finds an aligned position. This new aligned position is given as the argument to a function **getbits1.m** which will take this argument as a starting position and the length of **source.txt** as an ending position and will transfer all the aligned data in **mainsource.txt.** Then it will start channel splitting by taking the data from **mainsource.txt.**

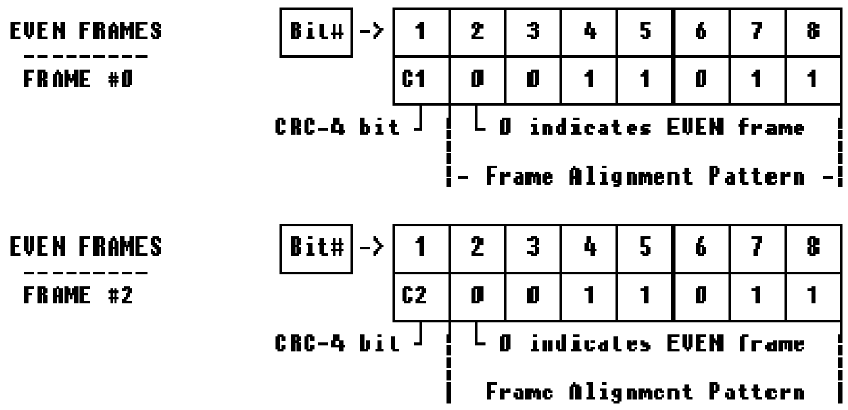
Similar is the case for checking the super frame alignment. In it the multi frame alignment signal is also checked which comes in TS16 of Frame 0 of every super frame. If these conditions are true, the channel splitting algorithm runs, which splits the data into the channels. Otherwise if either of these conditions are false, then we collect 4096 bits of data i.e. receive a super frame from the server. The alignment function **tfallign.m** will run again with take the last position of the previously aligned data as input and will return a new aligned position. If it fails to find an aligned position then it will again collect one more super frame i.e. 4096 bits from the server and this will continue until it finds an aligned position. Then it will split channels from the new alignment position till the end of the data.

After doing channel splitting it draws the graphs for all 32 channels on the main GUI. Graphs will be made in time domain by default. If the user clicks on the frequency domain button on the GUI before frame alignment, the graphs will be displayed in frequency domain.

5.5.3 Frame Alignment

Suppose when you connect the E1 cable to the system, bit 3 of frame 17 enters into the computer. Now there must be some method of letting the computer know the start of the frame. A pattern exists to determine this. Every even frame's TS0 has an 8bit signal pattern **X0011011**. This pattern appears again after 512 bits. Based on this the frame alignment and recognition algorithm works.

A frame alignment signal of 0x9B (155 decimal, **10011011** binary) or 0x1B (27 decimal, **00011011** binary) appears in TS0 of every even frame. Moreover, TS16 in frame 0 of super frame has the multi frame alignment signal of **0000** binary. This is shown in figure 5-2. So checking these two signals will lead us to detect the start of every super frame.



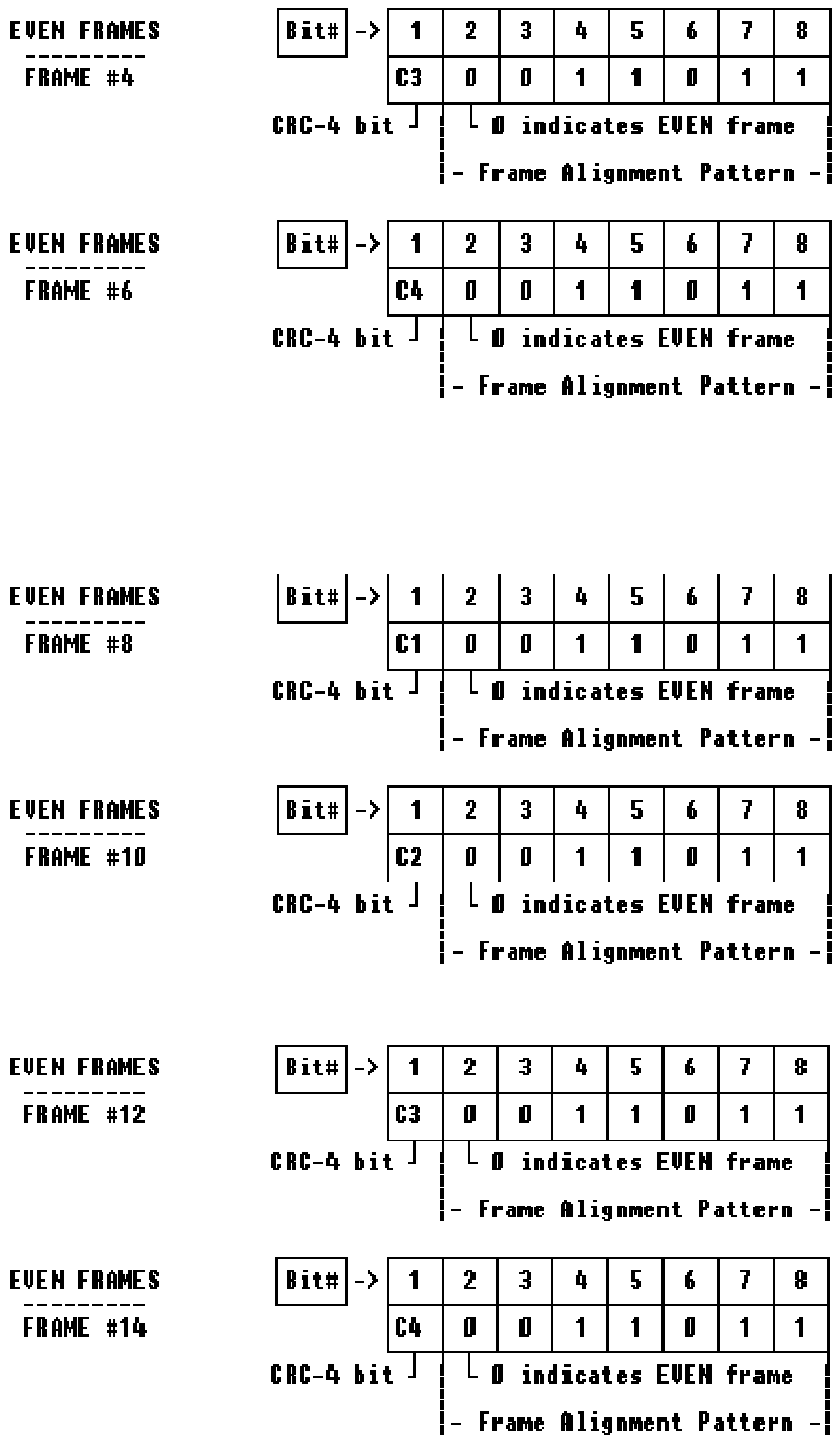


Figure 5-2 Even Frame Alignment Pattern

CRC bits have been explained earlier in section 3.3

This frame alignment pattern can also come in the data. To make sure, that the same pattern in the data is not misunderstood as the frame alignment signal (FAS), a probability check is used. A frame alignment signal (FAS) repeats itself after every 512 bits as mentioned earlier. If such a pattern is observed in data, it has a low probability of repeating itself after 512 bits the second time. Furthermore, it has much less probability of repeating itself the third time after the next 512 bits. So the probability check makes sure, that the Frame alignment signal pattern is repeated at least twice after its occurrence for the first time.

The frame alignment function is **tfallign.m.** It runs only when 4096 bits i.e. one super frame of binary stream have been stored in **source.txt**. The control signals i.e. the frame alignment signal for both TS0 and TS16 are defined and thus converted into numeric values for comparison. This function runs until it finds the starting position of a super frame. If this function fails to find the aligned position, it return 0 and the program will again get one super frame from the server i.e. 4096 bits **tfallign.m** starts with the first position i.e. variable i=1 and stores the next seven values in an array temp. This can be understood by looking at the table 5-4

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
i	i+1	i+2	i+3	i+4	i+5	i+6	i+7

Table 5-4 Distribution in Variable Temp

Then temp compares itself with the control signals of TS0 i.e. **X0011011**. If it matches then it increments i by 512, as the control signal appear after every 512 bits in an even frame. Then it will store the next 8 bits for the next even frame and compare them with the control signals of TS0. This is repeated one more time i.e. the third time incase the second check is true.

Now if in the first match, the comparison is false. i will be incremented to the next position and will check the next 8 values as described above.

Similarly if the second of third match yields a false, then it will return to start of loop and again increment i by 1. This will keep going until FAS is found. This logic is explained below with the help of a flow diagram.

If FAS is found in the binary data three times, then it will also check the control signal for TS16 i.e. **0000** by incrementing i by 120 bits. Since, it appears after 121 bits of the starting position of super frame. Now i is returned as the required starting position by this function. The returned value describes the position at which the frame alignment starts i.e. the first bit of TS0 of frame 0 in the super frame.

Alignment algorithm can be seen in Table 5-5 given in Appendix C.

5.5.4 Channel Splitting:

The channel splitting function will always be called when there is sufficient amount of data present in **mainsource.txt** that can be equally distributed among 32 channels.

The channel splitting function **chspliter.m** takes an argument as a starting position, and takes the ending position as the end of file **mainsource.txt.** From the starting position onwards it will take 8 bits

- Convert them into numeric values,
- Converts them into decimal values and
- Will assign them to the channels serially.

In this way it will split all the data among the 32 channels. Data is converted into decimal because of ease of use, when plotting graphs. Note that the channel splitting function will be only called whenever there is sufficient amount of data, capable of being distributed among 32 channels. This channel splitting algorithm returns the ending position i.e. length of the file until which it has split the data, so that the next time this function is called; it will start from returned position +1 in the file **mainsource.txt.**

A slight idea of the main algorithm is given in Table 5-6.

```
 z;
tt=[];
tt = [A(z) A(z+1) A(z+2) A(z+3) A(z+4) A(z+5) A(z+6) A(z+7)];
mt=[];
for hh=1:length(tt)
  mt=[ mt num2str(tt(hh))];
end
mt;
tt=bin2dec(mt);
ch1 = cat(2,ch1,tt);

z=z+8;
tt=[];
tt = [A(z) A(z+1) A(z+2) A(z+3) A(z+4) A(z+5) A(z+6) A(z+7)];
mt=[];
for hh=1:length(tt)
  mt=[ mt num2str(tt(hh))];
end
mt;
tt=bin2dec(mt);
ch2 = cat(2,ch2,tt);
………………..
And so on… (for each channel)
```

Table 5-6 Channel Splitting

5.5.5 CRC Finder:

As mentioned earlier, a 4-bit Cyclic Redundancy Check (CRC) is sent in E1 data. The CRC bits in Frames 0, 2, 4, and 6 provide error detection for the previously transmitted/received SUB-MULTIFRAME (Frames 0 through 7). The CRC bits in Frames 8, 10, 12, and 14 provide error detection for the previously transmitted/received SUB-MULTIFRAME (Frames 8 through 15).

NOTE: Refer to the section 3.3 for more information on CRC.

The **crc_finder.m** will be called in **chspliter.m** whenever the length of channel 1 will be greater than 16 i.e. at least one super frame is present in **mainsource.txt.** This function receives an argument as the starting position in the channel of the super frame as an input, puts the next 16 values i.e. super frame into a variable ch1_b. For each of the 16 values, it will take the first bit which is the CRC bit, and will store them into variable c1, c2…c8 respectively. The CRC bits c1, c2, c3, c4 correspond to the CRC bits of sub multi frame (SMF) 1. And the bit c5, c6, c7, c8 correspond to CRC bits of sub multi frame (SMF) 2.

Then it is matches the crc of SMF (N) with SMF (N+1) bit by bit, as both should be equal. This all is explained above in the main introduction under the heading of CRC. If the match isn't equal it shows an indication on the main GUI stating **"CRC BITS MISMATCH".** This function **crc_finder.m** returns the starting position of the next super frame, so when it is called next time, it will extract CRC bits from next super frame.

Whenever the **Crc bits info** button is clicked on the menu bar, the control goes to **crcinfogui.m.** In this function, the particular bits for a super frame are retrieved from variables c1, c2, c3, c4…c8 and are displayed.

Crcfinder.m Algorithm is given in Table 5-7 given in Appendix C.

5.5.6 Signaling Information:

CAS stands for Channel Associated Signaling. Examples are FXS loop start and E&M wink start. These protocols provide information such as the number that was called, and what state the call is in.

When Timeslot 16 of the E1 frame is used for Channel Associated Signaling purposes, Frame 0 contains information that is used by the receiver to identify the incoming frame. Specifically, this pattern in Timeslot 0, Frame 0 is called the Multi Frame Alignment Signal (MFAS). Frames 0 is shown in Table 5-8.

5.5.6.1 Frame:0

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
0	0	0	0	X	Y	X	X

Table 5-8 Frame 0

X = Spare Bits, set to 1 if not used.

Y = Yellow Alarm (Loss of Multi Frame Alignment Signal)

(0 = Normal | 1 = Loss of MFAS)

5.5.6.2 Frames 1-15: Signaling Bits

When Channel Associated Signaling is used, Timeslot 16 of Frames 1-15 is used to convey the state of the A, B, C and D signaling bits. Some notes:

- The ABCD state of 0000 is not allowed. If all bits in Timeslot 16 are 0, a Loss of Multi Frame Alignment Signal can be assumed.
- When A-Bit only signaling is used (No BCD bits), the BCD bits should be fixed at: B=1, C=0, D=1 (101). All frames shown in Table 5-9

Frame:1

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Ch1-A	Ch1-B	Ch1-C	Ch1-D	Ch16-A	Ch16-B	Ch16-C	Ch16-D

Frame:2

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Ch2-A	Ch2-B	Ch2-C	Ch2-D	Ch17-A	Ch17-B	Ch17-C	Ch17-D

Frame:3

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Ch3-A	Ch3-B	Ch3-C	Ch3-D	Ch18-A	Ch18-B	Ch18-C	Ch18-D

……………….

Frame:15

Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Ch15-A	Ch15-B	Ch15-C	Ch15-D	Ch30-A	Ch30-B	Ch30-C	Ch30-D

Table 5-9 Frame 1-15

Function **siginfo.m** is called in **chspliter.m** which extracts the signaling information for each channel in the super frame. This function will be called whenever the length of channel 1 will be greater than 16 i.e. at least one super frame is present in **mainsource.txt.** This function receives an argument as the starting position in the channel of the super frame as an input, puts the next 16 values i.e. super frame into a variable ch17_b. The first value in ch17_b will correspond to TS16 of frame 0, the second to TS16 of frame 1 and so on. As explained above, channel 2 and channel 18 signaling bits will be extracted from the second value of ch17_b. Similarly channel 3 and channel 19 signaling bits will be extracted from third value of ch_17b and so on. This function **siginfo.m** returns the starting position of the next super frame, so when it is called next time, it will extract signaling bits from next super frame.

Whenever the **Signaling info** button is clicked on the menu bar, the control goes to **siginfogui.m,** which will call a function **convert.m.** In this function, the particular signaling bits for a super frame are retrieved from variables ch1_siginfo, ch2_siginfo, ch3_siginfo …. ch30_siginfo and are displayed.

5.5.7 Graphical Analysis Functions:

When **channelinfo** button is clicked on the main GUI it prompts a dialog box, which requires the user to input a certain channel number between 1 & 32. There are two conditions to be checked before giving information for a certain channel. First it needs to make sure that frame alignment has been done, and there is sufficient data to analyze a single channel. Frame alignment requires that at least 4096 bits of E1 binary stream have been received. If channel info is required before frame alignment, an error dialog will be displayed.

Secondly, once sufficient data is present, it checks if the input is outside the range 1-32. If yes, an error dialog box is displayed. Otherwise if a correct value is entered within the range; it calls **ftransform.m** which displays all type of graphical analysis for that particular channel number which was entered.

When the function **ftransform.m** is called, it displays nine different kinds of graphs serially through the **choose.m** function.

1. Time vs. Amplitude
2. Time vs. Amplitude last 50 values
3. Fourier Coefficients in the complex plane
4. Frequency vs. Power
5. Stem plotting of Frequency vs. Power
6. Period vs. Power
7. Highlighting the Max Power at a particular Period
8. Power Spectral density
9. Power Spectral density of last 50 values

Please refer code in Appendix A for further explanation for each of the given graphs.

5.5.8 Plottd.m:

Whenever user clicks on **Time Domain** button in GUI, or after getting 256 bits, the function **plottd.m** is called. It consists of all the functions for drawing graphs in time domain for every channel. In case the channel has more than 20 values, it will display the last 20 values in the graphical plot. Algorithm is given in Table 5-10.

```
global ch1;

lgh=length(ch1);

qs=1;

cchh1=[];

if lgh > 20

  for ia=(lgh-20) : lgh

       cchh1(qs)=ch1(ia);

       qs=qs+1;

   end

   plot1 = plot(cchh1,'Parent',handles.axes1);

else

   plot1 = plot(ch1,'Parent',handles.axes1);

end
```

Table 5-10 Time Graph Algorithm

5.5.9 Plotfd.m:

Whenever user clicks on **Frequency Domain** button in GUI, or after getting 256 bits, the function **plotfd.m** is called. It consists of all the functions for drawing graphs in frequency domain for every channel. In case the channel has more than 20 values, it will display the last 20 values in the graphical plot.

Algorithm is given in Table 5-11 given in Appendix C.

5.6 Features of E1 Analyzer

- Aligns data to get the start of the super frame
- Detects if alignment goes out, and realigns again
- Multi frame alignment
- Separate monitoring of each channel.
- Graphical Analysis of all the channels
    - Time Domain is presented using oscilloscope display.
    - Frequency Domain is presented using bi spectral analysis.
- Graphical analysis of each channel can monitor the signal
    - Amplitude (to determine stability, signal susceptibility and strength of the signal)
    - Power (to determine range of signal or power surge)
    - Frequency coefficients (to determine the signal stability whether it is stable, marginally stable, or unstable)
- Checks CRC for data validation.
- Displays CRC information.
- Displays Signaling information (a b c d bits) for depicting the state of all channels.

6 Conclusions

6.1 Results

An E1 stream is generated using E1 generator. This generated E1 stream is tested through an E1 analyzer made in MATLAB. This analyzer performs real time analysis on the E1 data, in the form of Graphical analysis. Additionally it also checks CRC bits and further displays signaling information.

The graphical analysis consists of nine different kinds of graphs in both time and frequency domain. The time domain graphs are simple time vs. amplitude graphs. The frequency domain graphs are obtained by applying fast Fourier transform (fft) on the data.

CRC bits are being checked in the analyzer which is consistent with the real e1 data. If there is a mismatch in these bits, between data being generated, and the data being received, an error message is generated.

An indicator is there to determine if the frame alignment falls out. This is tested and as a result, if alignment pattern is not consistent, an error is detected.

Yellow alarms are also checked in the algorithm. If there is a yellow alarm in the data, it is checked and displayed. This has been tested in correspondence with the data generated.

6.2 Future Work

The hardware portion consists of the circuit up to the decoding of the real E1 stream. Because of shortage of time, the interfacing portion is incomplete. The primary task in the future is to interface the circuit with the computer via the USB port.

Additionally there is an algorithm developed which can operate on E1 data stored in a source file, and it is capable of sending data to the E1 analyzer. There is a need for receiving data from the port, so it could be further sent to the E1 analyzer.

Some additional features can be included in the software portion according to the requirements of the next project.

Based on this project, multiple projects can be developed, such as voice recognition of channels, Speaker identification, speech identification etc.

In speaker identification, voices for different persons could be stored a library. The voice of these people could be monitored on the voice channels, and whenever a speaker appears on the line, the channel could be high-lighted/recorded.

For speech identification, military sensitive words, such as 'suicide', 'war', ' attack' etc can also be stored in a library, speech identification on the voice channels will then recognize these words and identify the channel.

APPENDIX A

IC NUMBERS

IC NUMBERS

IC_1 --------------	6C32
IC_2 -------------	474
IC_3 --------------	404
IC_4 --------------	4179
IC_5 --------------	421
IC_6 --------------	474
IC_7 --------------	432
IC_8 --------------	4179

Voltage regulator	7805
Clock	17555
Transformation	bal-un

R_1 ------------	100Ω
R_2 ------------	4.7K
R_3 ------------	10K
R_4 ------------	100Ω
R_5 ------------	10K
R_6 -------------	4.7K
R_7 -------------	8.2K

APPENDIX B

SCHEMATIC DIAGRAMS

APPENDIX C

SOURCE CODE

E1 Generator

```
void E1Init(void)
{

     srand((unsigned)time(NULL));
     pucE1_Data = ucE1DataArray;
}

unsigned char crcNaive(unsigned char const message)
{
    unsigned char  remainder;
    unsigned char bit;

    remainder = message;



  for (bit=8; bit>0; --bit)
    {

        if (remainder & 0x80)
        {
           remainder ^= POLYNOMIAL;
        }

        remainder = (remainder << 1);
       }


return (remainder >> 4);

}

void d2b(int x)
{
   int j;
   char bin[4];
   int g=0,y=3;

   while(x>1)
   {
                  bin[g]= x%2;
                  x=x/2;
                  g++;
    }
   bin[g]=x;
   g++;
   bin[g]='\0';
    for(j=0; j<4; j++)
   {
           bin1[j]=bin[y];
           y--;
    }

    bin1[4]='\0';
}
int b2d(char str[])
{
           int x,u;
           int val[8];
           for(u=0; u<8; u++)
           {
                  val[u]=(int)str[u];

                  if (val[u]==49)
                  {
                            val[u]=1;
                  }
                  else if (val[u]==48)
                  {
                            val[u]=0;
                  }

           }

           x=  ((val[3]*1) + (val[2]* 2) + (val[1]*4) + (val[0]*8) );


   return x;

}

void GetE1_Data(void)
{
           z= (rand() & 0x00FF);
           if(z<16)
           {
                  z= (rand() & 0x00FF);
           }

     *(pucE1_Data + uiHeadPtr) = z;
     uiHeadPtr++;




     if(uiHeadPtr == E1_BUFFER_SIZE)
     {
         uiHeadPtr = 0;
     }
}

BOOL ReadE1_Data(unsigned char * pucDataBuff)
{

     int o=0;
     static unsigned char ucByteCount = 0;

     unsigned char ucIndex;

     if(uiTailPtr < uiHeadPtr)
     {

         if((uiHeadPtr - uiTailPtr) < 4)
             return FALSE;
     }
     else
     {
         if((E1_BUFFER_SIZE - uiTailPtr + uiHeadPtr) < 4)
             return FALSE;
     }

     for(ucIndex = 0; ucIndex < 4; ucIndex++)
     {
         *(pucDataBuff + ucIndex) = *(pucE1_Data+ ((uiTailPtr + ucIndex) %
                                             E1_BUFFER_SIZE));
     }

     uiTailPtr = (uiTailPtr + 4) % E1_BUFFER_SIZE;

     if(ucByteCount == 0 || ucByteCount == 64 )
     {

                     if(j==4)
                     {
                            printf("\tTRUE");
                            arr[j]='\0';
                            j=0;

                            printf("\tARR:");
                            for(o=0; o<4; o++)
                            {
                              printf("%d",arr[o]);
                            }printf("\t");

                            dec=b2d(arr);

                            hex=dec;

                            if(hex == 10)
                                    hex=0xA;
                            if(hex == 11)
                                    hex=0xB;
                            if(hex == 12)
                                    hex=0xC;
                            if(hex == 13)
                                    hex=0xD;
                            if(hex == 14)
                                    hex=0xE;
                            if(hex == 15)
                               hex=0xF;

                      msg=hex;
                      printf("MSG: %X",msg);
                      res=crcNaive(msg);
                      printf("CRC: %d",res);
                      d2b(res);

                      printf("\tBIN: ");
                      for(o=0; o<4; o++)
                      {
                              printf("%d",bin1[o]);
                      }


                     }
                      z= bin1[j];
                      if(z==1)
                              z=155;
                      else
                              z=27;
```

```
                    *pucDataBuff = z;
                       ucByteCount = 0;
                       printf("\tTRUE");

                    if(z==155)
                                arr[j]='1';
                        else
                                arr[j]='0';
                        j=j+1;

}

if((*(pucDataBuff + 3)%200) > 170 )
           {
                        z= (rand() & 0x00FF);
                         if(z>127)
                                z=155;
                         else
                                z=27;

        *(pucDataBuff + 2) = z;
           }

    ucByteCount += ucIndex;

    return TRUE;
}
```

Sending E1 data to Client

```
    while(1)
    {
        GetE1_Data();

        if(ReadE1_Data(ucDataBuff))
                    {

                                if(j==0)
                                {


                                        if(i==3)
                                          {
                                                ucDataBuff[3]=0xF;
                                                i=0;
```

```
                              j=1;
                        }
                  }



                  if(j==1)
                  {
                        if(i==128)
                        {
                              ucDataBuff[3]=0xF;
                               i=0;
                        }
                  }


                  printf("\r\n%d %d %d %d", ucDataBuff[0],
               ucDataBuff[1], ucDataBuff[2], ucDataBuff[3]);

                  send(xClientSocket, ucDataBuff, 4, 0);
                  i=i+1;
                  Sleep(100);
            }
            Sleep(50);
   }
}
```

Table 5-1 E1 Generator Algorithm

E1 Retriever

```
   for(u=0; u<8; u++)
      {
            val[u]=(int)str[u];

            if (val[u]==49)
            {
                  val[u]=1;
            }
            else if (val[u]==48)
            {
                  val[u]=0;
            }
            //cout<<val[u]<<" ";
            }

            x=  ((val[7]*1) + (val[6]* 2) + (val[5]*4) + (val[4]*8) + (val[3]*16) +
                  (val[2]* 32) + (val[1]*64) + (val[0]* 128) );

            return x;
}

int get8bits(void)
{
   char e1[8];
   char str1[8];
   int d1,i;

   FILE * pFile;
   char string [32];

   pFile = fopen ("e1data.txt" , "r");
   if (pFile == NULL) perror ("Error opening file");
   else {

     fseek ( pFile , pos , SEEK_SET );
            pos=pos+8;
            fgets (string , 9 , pFile);
          fclose (pFile);
   }

   strcpy (e1,string);
   for(i=0; i<8; i++)
   {
             str1[i]=e1[i];

   }
  str1[8]='\0';

   d1=b2d(str1);

   return d1;
}



void GetE1_Data(void)
{
     int z;
      z= get8bits();
```

```
    *(pucE1_Data + uiHeadPtr) = z;
    uiHeadPtr++;

    if(uiHeadPtr == E1_BUFFER_SIZE)
    {
        uiHeadPtr = 0;
    }
}

BOOL ReadE1_Data(unsigned char * pucDataBuff)
{

    static unsigned char ucByteCount = 0;
    unsigned char ucIndex;

    if(uiTailPtr < uiHeadPtr)
    {

        if((uiHeadPtr - uiTailPtr) < 4)
            return FALSE;
    }
    else
    {

        if((E1_BUFFER_SIZE - uiTailPtr + uiHeadPtr) < 4)
            return FALSE;
    }

    for(ucIndex = 0; ucIndex < 4; ucIndex++)
    {

        *(pucDataBuff + ucIndex) = *(pucE1_Data+ ((uiTailPtr + ucIndex) %
         E1_BUFFER_SIZE));
    }

    uiTailPtr = (uiTailPtr + 4) % E1_BUFFER_SIZE;
     ucByteCount += ucIndex;

    return TRUE;
}
```

Sending it to Client

```
while(1)
    {
        GetE1_Data();

        if(ReadE1_Data(ucDataBuff))
                {

                        if(j==0)
                        {


                                if(i==3)
                                  {
                                        ucDataBuff[3]=0xF;
                                        i=0;
                                        j=1;
                                  }
                        }



                         if(j==1)
                        {
                                  if(i==128)
                                  {
                                        ucDataBuff[3]=0xF;
                                         i=0;
                                  }
                        }


                        printf("\r\n%d %d %d %d", ucDataBuff[0],
                    ucDataBuff[1], ucDataBuff[2], ucDataBuff[3]);

                        send(xClientSocket, ucDataBuff, 4, 0);
                        i=i+1;
                       Sleep(100);

 }
                Sleep(50);
    }
}
```

Table 5-2 E1 Retriever Algorithm

Frame Alignment

```
for qq=1:length(cs1)
  contsig1=[ contsig1 num2str(cs1(qq))];
end

for qq=1:length(cs2)
  contsig2=[ contsig2 num2str(cs2(qq))];
end

for qq=1:length(cs3)
  contsig3=[ contsig3 num2str(cs3(qq))];
end

while (i<length(A)-6)

    temp = [A(i) A(i+1) A(i+2) A(i+3) A(i+4) A(i+5) A(i+6) A(i+7)];
    mmat=[];
    for ww=1:length(temp)
      mmat=[ mmat num2str(temp(ww))];
    end
    mmat;

     if(mmat == contsig1 | mmat == contsig2)

         display('MATCHED 1');
         i;
         z=i;

         i = i+512;
         if(i>length(A)-6)
            break;
```

```
end
temp = [A(i) A(i+1) A(i+2) A(i+3) A(i+4) A(i+5) A(i+6) A(i+7)];
mmat=[];
for ee=1:length(temp)
mmat=[ mmat num2str(temp(ee))];
end
mmat;
 if(mmat == contsig1 | mmat == contsig2)
        display('MATCHED 2');
        i;
        z2=i;
        i = i+512;
        if(i>length(A)-6)
           break;
        end
      temp = [A(i) A(i+1) A(i+2) A(i+3) A(i+4) A(i+5) A(i+6)
                  A(i+7)];
      mmat=[];
      for rr=1:length(temp)
      mmat=[ mmat num2str(temp(rr))];
      end


       mmat;
       if(mmat == contsig1 | mmat == contsig2)
           display('MATCHED 3');
           i;
           d = z + 120;
           temp = [A(d) A(d+1) A(d+2) A(d+3) A(d+4) A(d+5) A(d+6)
                       A(d+7)];
          mmat=[];
          for rr=1:length(temp)
```

```
                    mmat=[ mmat num2str(temp(rr))];
               end

                if(mmat == contsig3)
                    display('TS16 MATCHED');
                    reqposition = z ;
                    break;
                else
                    i=z;

                end

              else
                 i=z;
              end

          else
             i=z;
         end

end
     i=i+1;
end



if(reqposition ~=0)
    set(han.aout,'Visible','Off');
    set(han.adata,'Visible','Off');
    getbits1(reqposition,han);

else
```

```
   disp('FAILED REQPOSITION');
   x=0;
   y=0;
   reqposition=0;
end
```

Table 5-5 Frame Alignment Algorithm

CRC Finder

```
if(onetym==1)

   smf1_crc=[c1(v) c2(v) c3(v) c4(v)];
   smf2_crc=[c5(v) c6(v) c7(v) c8(v)];

   smf1_crc=[smf1_crc '0000'];

   rem=dec2bin(poly_div(smf1_crc));

   if(rem~=smf2_crc)
      set(hh.crcout,'Visible','On');
   else
      disp('crc info matched');
   end

   smf2_crc=[smf2_crc '0000'];
   rem=dec2bin(poly_div(smf2_crc));
   tobechecked=rem;

   sz=size(tobechecked);
    diff=4-sz(2);
    if(sz(2)< 8)
      for ad=1:diff
        tobechecked= strcat('0',tobechecked);
      end
    end
end

if(onetym==0)

   smf1_crc=[c1(v) c2(v) c3(v) c4(v)];
   smf2_crc=[c5(v) c6(v) c7(v) c8(v)];

   if(tobechecked~=smf1_crc)
      set(hh.crcout,'Visible','On');
   else
      disp('crc info matched');
   end
  smf1_crc=[smf1_crc '0000'];

   rem=dec2bin(poly_div(smf1_crc));

   if(rem~=smf2_crc)
      set(hh.crcout,'Visible','On');
   else
      disp('crc info matched');
   end

   smf2_crc=[smf2_crc '0000'];
   rem=dec2bin(poly_div(smf2_crc));
   tobechecked=rem;
  sz=size(tobechecked);
  diff=4-sz(2);
  if(sz(2)< 8)
     for ad=1:diff
       tobechecked= strcat('0',tobechecked);
     end
  end
end

onetym=0;
qs=f+16;
```

Table 5-7 CRC Finder Algorithm

Frequency Graph

Fourier Coefficients in the Complex Plane

```
Y = fft(channel_no);

Y(1)=[]

plot(Y,'ro')
```

Periodogram

```
Y = fft(channel_no);

n=length(Y);

power = abs(Y(1:floor(n/2))).^2;

nyquist = 1/2;
```

```
freq = (1:n/2)/(n/2)*nyquist;

plot(freq,power)
```

Period vs Power

```
Y = fft(channel_no);

n=length(Y);

power = abs(Y(1:floor(n/2))).^2;

nyquist = 1/2;

freq = (1:n/2)/(n/2)*nyquist;

period=1./freq;

plot(period,power);
```

Power Spectral Density

```
Y = fft(y,256);

Pyy = Y.*conj(Y)/256;

f = 1000/256*(0:127);

plot(f,Pyy(1:128))
```

Table 5-11 Frequency Graph Algorithm

BIBLIOGRAPHY

Bibliography:

[1] ITU-T Recommendation G.701, *'Vocabulary of digital transmission and multiplexing, and pulse code modulation (PCM) terms'*, approved in 1993.

[2] ITU-T Recommendation G.702, *'Digital hierarchy bit rates'*, 1998.

[3] ITU-T Recommendation G.703, 'Physical/electrical characteristics of hierarchical digital interfaces '2001

[4] ITU-T Recommendation G.704, *'Synchronous frame structuresused in primary and secondary hierarchical levels'*, 1998

[5] ITU-T Recommendation G.711, *'Pulse code modulation (PCM) of voice frequencies'*, 1998.

[6] ITU-T Recommendation G.732, *'Characteristics of primary PCM multiplex equipment operating at 2048 kbits/s'*, 1988.

[7] Grahame Smillie, *'Analogue and Digital Communication Techniques'*, pages 100-124, Published by Arnold 1999.

[8] 2.048 Mbps technology basics and testing fundamentals

[9]*'E1 Digital Facilities'*, http://www.cisco.com/warp/public/788/signalling/e1_r2_sig.html

[10] *'E-carrier'*,http://en.wikipedia.org/wiki/E-carrier

[11] *'Fourier series'*, *http://en.wikipedia.org/wiki/Fourier_series*

[12] *'Matlab'* ,http://www.mathworks.com/products/matlab/

[13] Extension of 2MB stream on RL 421. Degree project 1999.