EMBEDDED SOFTWARE USING C/C++ FOR DIGITAL SET-TOP BOX WITH CONDITIONAL ACCESS FOR PAY TV CHANNELS

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

SUJEET LOHAN

ER & DCI
NOIDA

IIT Roorkee – ER&DCI, Noida
C-56/1, "Anusandhan Bhawan"
Sector 62, Noida - 201 307

FEBRUARY, 2003

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled **"EMBEDDED SOFTWARE USING C/C++ for DIGITAL SET -TOP BOX with CONDITIONAL ACCESS for PAY-TV CHANNELS"**, in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT, Roorkee – ER&DCI Campus, Noida,** is an authentic record of my own work carried out during the period from August 2002 to February, 2003 under the guidance of **Dr. P.C. Jain**, GM, R& D Division , Himachal Futuristic Communication Ltd., Gurgaon.

The matter embodied in this dissertation has not been submitted by me for award of any other degree or diploma

Date: 26\02\03

Place: Noida **(SUJEET LOHAN)**

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

26-02-03

Prem Chand Jain

Co-Guide:
Mr. Munish Kumar, Project Engineer
ER & DCI, Noida

Project Guide :
Dr. P.C. Jain , General Manager
HFCL (R& D) , Gurgaon

ACKNOWLEDGEMENT

The timely completion of the work carried out in this dissertation report would not have been possible without the cooperation and guidance of team members of Set-top Box project. In first place, I am indebted to my project guide, **Dr. P. C. Jain**, for his consistent support and valuable guidance. I am also grateful to **Mr. V. N. Shukla** , Course Coordinator, ER&DCI for his useful suggestions and encouraging me to take up this project. I am also thankful to my co-guide Mr. Munish Kumar,Project Engineer, ER & DCI for all the cooperation and help extended by him in the successful completion of this project.

Also , my sincere thanks to Mr. Sushil Dutt, and Mr. Sarfaraz Ahmed , both Sr. Engineer in R & D Division of HFCL, for fruitful discussions and providing valuable information which could led to timely completion of the project.

Finally, a vote of thanks to all those who directly or indirectly, in some manner or the other , contributed towards the project work.

(SUJEET LOHAN)
Enroll. No. - 019050

CONTENTS

ABSTRACT

Today , a digital TV requires a Set-Top Box (STB) i.e. box which is on top of analog TV which is used to decode and tune digital signals, and converts them to a format that is understood by analog TV. The broadcasters and service providers are interacting with their viewers on many levels, offering them greater program choice than ever before. Additionally, the development of a security system provides them with unprecedented control over what they watch and when.

A CA system is best described as a virtual gateway that allows viewers to access a new world of digital services. The main goal of any CA system is to control subscriber's access to digital TV pay services and secure the service providers revenue streams. Consequently, only customers that have a valid contract with the network operator can access a particular service. CA System from Conax which is a CA provider has been implemented in the Set-top Box (STB) of HFCL. This is first part of the project carried out at HFCL.

To operate STB in a easy and interactive manner , a sort of Graphic User Interface (GUI) has been developed using Fujitsu,s API library functions. The main purpose of OSD is to make handling of Set –top Box (STB) convenient and interactive for the user. Various menus are popped up on TV screen and user can choose and view his favorite channel from the channel list. Also he can customize , set and adjust the various parameters from personal preferences such as password etc.

Next part is that of extraction of information from Transport Stream which is based on systems part of the **Recommendation/International Standard ISO/IEC ISO/IEC 13818-1: 1994(E) ITU-T Rec. H.222.0 (1995 E)** deals with the combination of one or more elementary streams of video and audio, as well as other data, into single or multiple streams which are suitable for storage or transmission. Systems coding is then carried out as per syntactic and semantic rules imposed by above specification and provides information to enable synchronized decoding of decoder buffers over a wide range of retrieval or receipt conditions. Both the Transport Stream and Program Stream defined in above standards provide coding syntax which is necessary and sufficient to synchronize the decoding and presentation of the video and audio information

Chapter 1

INTRODUCTION

1.1 Overview

I was involved in the team of set-top Box project as trainee in the R&D Division of HFCL,Gurgaon where I carried out my project work. One part of the project was to develop graphical user interface (GUI) also called On Screen Display (OSD) so that use of Set-topBox (STB) convenient and interactive. Various menus have been developed for this purpose using C/C++ which pop up on t TV screen when the user uses STB along with the analog TV. The user can make his/her choices while navigating through the menus. Also he/she can make his/her personal settings such as setting the passwords.
Second part was to extract the program specific information from Transport Stream(TS) which is MPEG encoded and to decode them and present them as a program.

1.2 Objective of the Dissertation

The main objective of the dissertation is to develop graphical user interface(GUI) called On Screen Manual (OSD) using C/C++ (main purpose of the OSD is to make the handling of Set-top Box (STB) easier so that it becomes simple to use) and to extract program specific information which is present in the form of tables in the MPEG encoded transport stream, decode and present them as the program. Also project objective also includes the study of Conditional Access Systems in the STB.

1.3 Scope of the work

This work is a part of the product making that is STB by HFCL, Gurgaon. This product as a whole will be used for use along with the Analog TV for Pay TV channels where the signals are encrypted and scrambled and needs descrambling and decryption. This can be extended to include such features as Electronic Program Guide(EPG), setting the favorite list of channels etc in the OSD part of the project and extraction of the date and time signals, event information table in TS

1.4 Organization of Thesis

The thesis is organized into eight chapters.
The first chapter deals with the basic concepts of conditional access system.
The chapter 2 to 5 cover all the classes developed and various functions used in making of menu's for OSD.
The chapter 6 tells various details about the packets and their format in Transport Stream.
The chapter 7 gives in depth coverage of various PSI tables.
The chapter 8 shows algorithm of extracting packets from TS and various mnemonics used.

Chapter 2

CONDITIONAL ACCESS SYSTEM

2.1 Introduction

Today , a digital TV requires a Set-Top Box (STB) i.e. box which is on top of analog TV which is used to decode and tune digital signals, and converts them to a format that is understood by analog TV. The broadcasters and service providers are interacting with their viewers on many levels, offering them greater program choice than ever before. Additionally, the development of a security system provides them with unprecedented control over what they watch and when. A CA system is best described as a virtual gateway that allows viewers to access a new world of digital services. The main goal of any CA system is to control subscriber's access to digital TV pay services and secure the service providers revenue streams. Consequently, only customers that have a valid contract with the network operator can access a particular service. The CA system is therefore, a vital aspect of digital TV business.

2.2 What is Conditional Access

A conditional access system (CAS) is a security module that comprises a combination of **scrambling** and **encryption** to prevent unauthorized reception. Encryption is the process of protecting the secret keys that are transmitted with a scrambled signal to enable the descrambler to work.

The scrambler key, called the **control word (CW)** must, of course, be sent to the receiver in encrypted form as an **entitlement control message (ECM).** The CA subsystem in the receiver will decrypt the control word only when authorized to do so; that authority is sent to the receiver in the form of an **entitlement management message (EMM).** Refer figure no. 2.1, pg 6

2.3 Parts of Conditional Access System

The system is primarily made up of three parts:

1. Signal Scrambling
2. Encryption of electronic "keys"
3. Subscriber Management System

2.3.1 Signal Scrambling

The MPEG-2 Audio, Video and data is fed to the scrambler unit along with scrambler key called the CW generated from control world generator. Scrambling involves modula-2 (Exclusive-OR) addition of the input data stream to a given Pseudo Random Bit Sequence (PRBS). CW is transmitted to the receiver in an encrypted form to prevent piracy attach on the CA system. A CW supplies a 64 bit word to the scrambler. This CW is encrypted to generate ECM. The ECM contains a program description and the actual CW. Refer figure no. 2.2 , pg 7.

The CA Sub System (CASS) in the STB will decrypt the control word only when authorized to do so. The entitlement to subscriber can be provided in the form of electronic **Smart Card (SC)** that is plugged into the STB. The authority to decrypt control world is sent to the STB in the form of EMM, which is subscriber specific. Consequently, the number of EMM that need to be sent over the broadband network is proportional to the number of subscribers that have registered on the network.

2.3.2 Subscriber Management System

The Subscriber Management System is a combination of hardware and software as well as human activities that help organize and operate the company business. The main goal of any SMS system is to ensure that subscribers view exactly what they pay for. The SMS is part of the Customer Management System (CMS) and includes:

Entitlement Control Messages

They carry program and service specific information , including the control words that are used by the smart card to decrypt the relevant program. How ever if a subscriber is not entitled to watch the program , then a signal is sent to the set-top box to indicate that this program has not been authorized for de-encryption.

Entitlement Management Messages

An Entitlement Management Message is used to carry authorization details which may be delivered electronically within the broadcast system. They are subscriber specific i.e. one EMM per subscriber.

Subscriber Authorization System

The main task of the SAS is to translate the requests coming from the SMS into authorization messages (EMMs). The EMM Injector receives EMM from SAS server, manages the EMM playout queues and feeds the EMMs into MPEG-2 multiplexer. They are sent to subscriber on a regular interval (e.g. every month) to renew subscription rights on the smart card.

2.4 CA Sub System at the receiving side

The receiving unit consists of STB, smart card and the remote controller. Smart card is a tamper free device that is issued to subscriber when they are registered. STB is used to descramble the viewing program, provide the human interface and handle other receiving details. Whenever a viewer wishes to see a scrambled service he has to ensure that a right smart card is inserted.Transport Stream (TS) contain Program Specific Information (PSI) and PSI tables (PAT, PMT, CAT). The Program Association Table (PAT) lists all current programs and indicates the Packet Identifier (PID) values of the Program MAP Table (PMTs). There is only one PAT but there are as many PMTs as programs. Each PMT gives information about the component streams (audio, video, data) and lists all parameters necessary for decoding the components.

There is one Conditional Access Table (CAT) per TS and it consists of a list of all the CA suppliers that work with programs found in the Transport Stream. The CAT carries the list of CA suppliers that provide services for programs found in the TS. A unique identifier called the CA system ID recognizes CA suppliers. The CAT also gives information about Entitlement Message Management (EMM) and PMT gives the information about the Entitlement Control Message (ECM). The TS carries packets corresponding to the scrambled signals but also simultaneously there will be packets from programs offered without access restrictions. It is at the STB where the packets are de-multiplexed and decrypted if required. Refer figure no. 2.3, pg 8.

A set-top box will demodulate the signals and will pass the Transport Stream to the de-multiplexer to acquire PSI data (PAT, CAT, PMT). EMM corresponding to the address of the smart card is passed to the security module as soon as it arrives. If the module decides that the user is in good standing, and he/she is allowed to watch (or download) certain programs in the TS, then decryption proceeds, and the descrambled TS is passed back to the host (STB) for decoding and display.

Whenever a new ECM arrives it is passed to the security module. ECM decryptor supplies the decrypted CW to the descrambler thus enabling it to descramble the service. There is a user interface module that display the messages like access denied or takes the input from the user input such as PIN code. Some well known CA systems includes:

1. Conax-CAS3 from conax
2. Cryptoworks from Philips.
3. Viaaccess from France Telecom.
4. Media Guard from Canal +

Following modules have been added to extend Conax CA in our STB:

1. Configure our de-multiplexer to extract CAT, ECM and EMM from the transport stream.
2. When ECM and EMM are acquired they are sent to the smart card in the form of commands, which are Conax proprietary, to the security module. In this case the security module is the smart card.
3. To communicate with the smart card ISO 7816-3 protocol is used.
4. Corresponding to command, a response is generally received from the smart card. This response is again Conax proprietary. Based on the response the required action is taken e.g. feeding the descrambler with the CW.
5. There is a user interface module that handles the communication between the security module and the user. For example user can use the dialogue to check the subscription status and event status. Similarly, security module can use it to enquire PIN code from the viewer.

2.4.1 Processing Sequence

The sequence of processing can be categorized into:

Initialization

It is performed either when the smart card is inserted in the smart card slot, or when the power is applied to the STB, or the communication with the smart card has fallen out of synchronization. Whenever smart card is reset, it sends Answer-To-Reset (ATR). ATR may contain interface information like clock rate, conversion factor, baud rate adjustment factor and extra guard time. After the link with smart card has been established the most recent ECM and EMM is sent to the smart card.

Access to program subject to conditional access

As described above the relevant CA desriptors and corresponding ECM PIDs are extracted from the PMT based on CA_SYS_ID then the filtering of ECMs start. Whenever a new ECM arrives it is sent to the security module. Here ECM is decrypted and checked for entitlement. If access is granted then CW is sent by the smart card as a response. This CW is fed to the descrambler.

EMM Processing

It is performed regardless of whether the current program is free-to-air or subject to conditional access. As soon as new EMM is acquired, it is sent to the smart card. Here the EMM is decrypted and then interpreted.

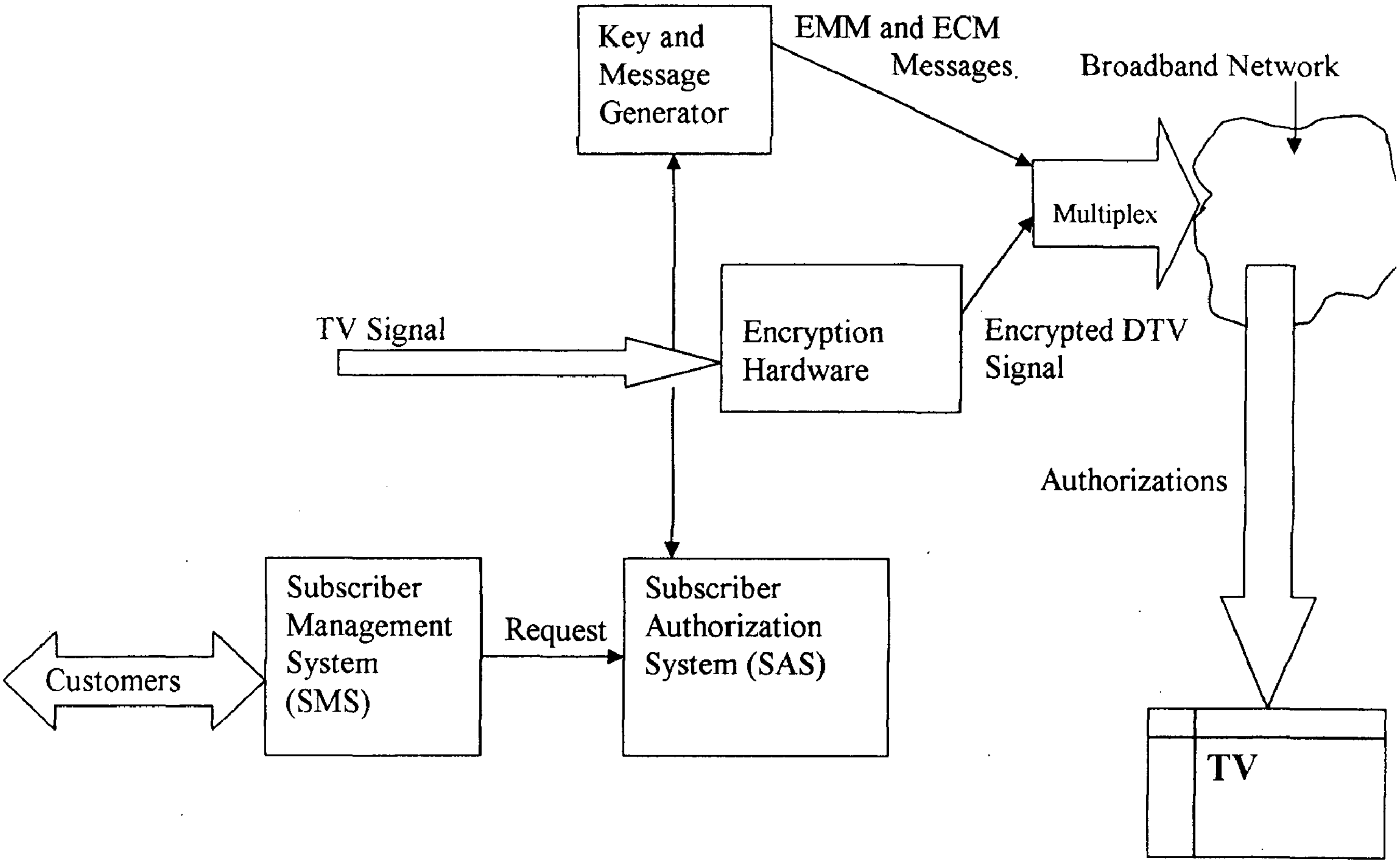


Figure No. 2.1 -Basic principle of an end-to-end Conditional Access System

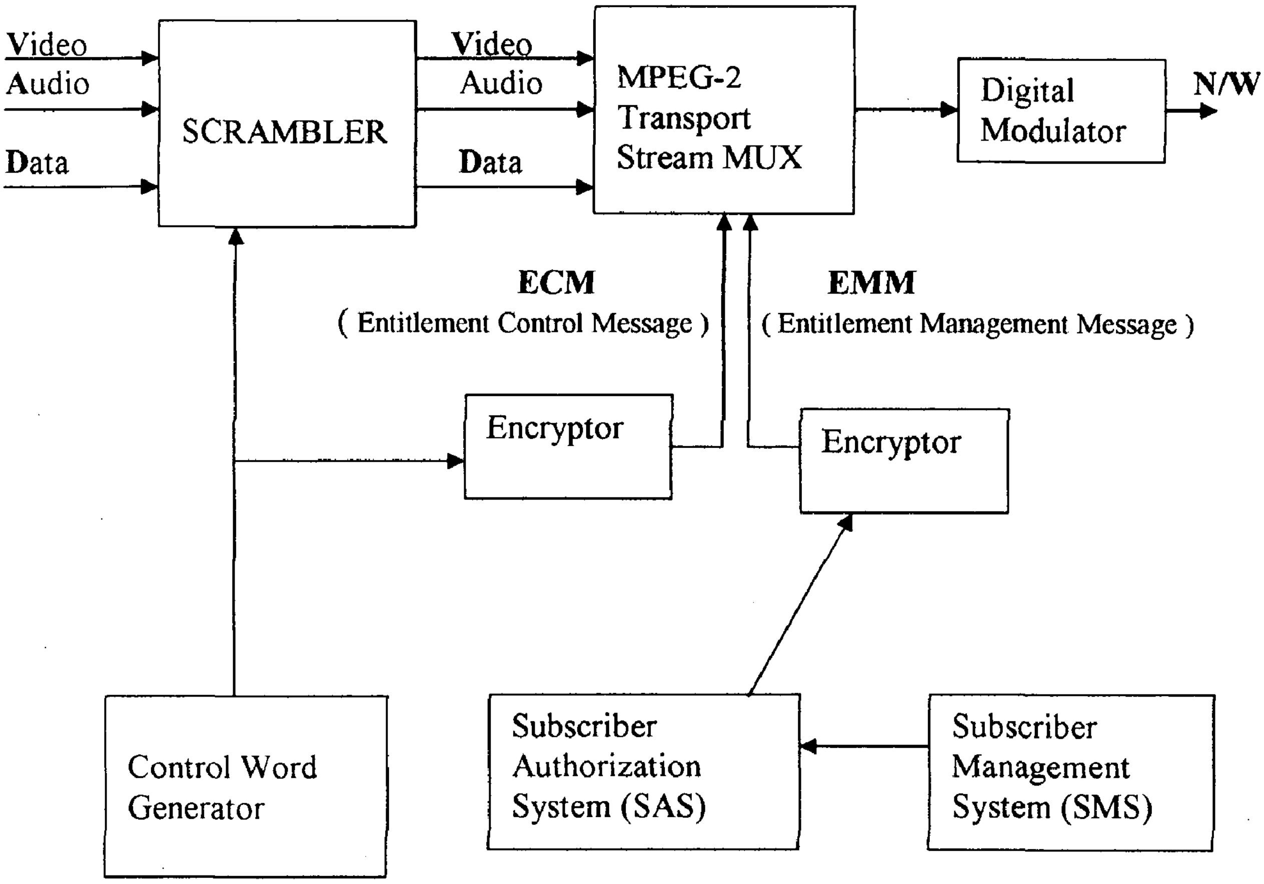

Video
Audio
Data
SCRAMBLER
Video
Audio
Data
MPEG-2 Transport Stream MUX
Digital Modulator
N/W
ECM
(Entitlement Control Message)
EMM
(Entitlement Management Message)
Encryptor
Encryptor
Control Word Generator
Subscriber Authorization System (SAS)
Subscriber Management System (SMS)

Figure No. 2.2 -General Conditional Access System (Head End)

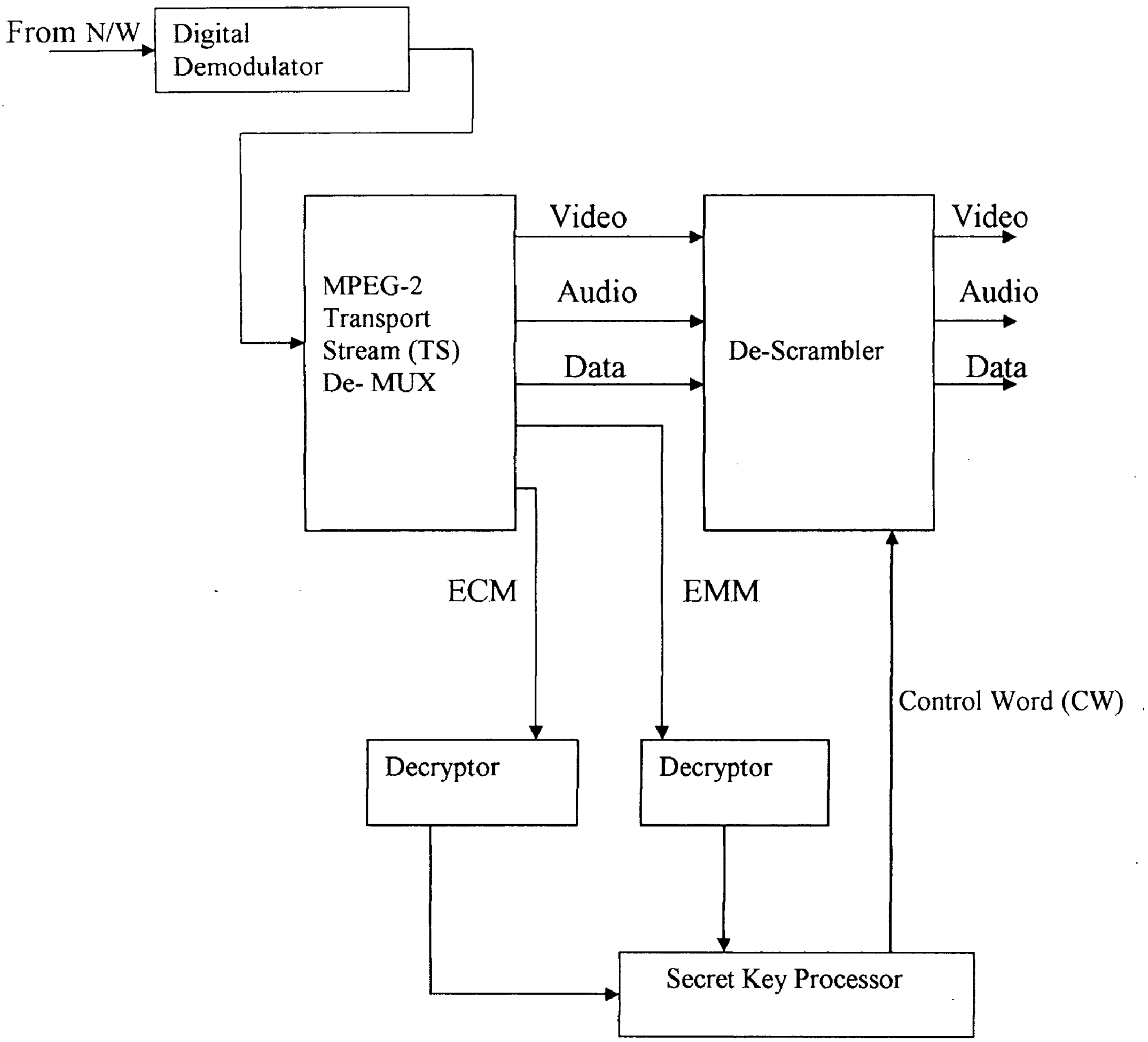


Figure No. 2.3 - Conditional Access System – Set-top Box (STB)

ON SCREEN DISPLAY

3.1 Purpose of OSD

This is a sort of Graphic User Interface (GUI). The main purpose of OSD is to make handling of Set –top Box (STB) convenient and interactive for the user. Various menus are popped up on TV screen and user can choose and view his favorite channel from the channel list. Also he can customize , set and adjust the various parameters from personal preferences such as password etc.

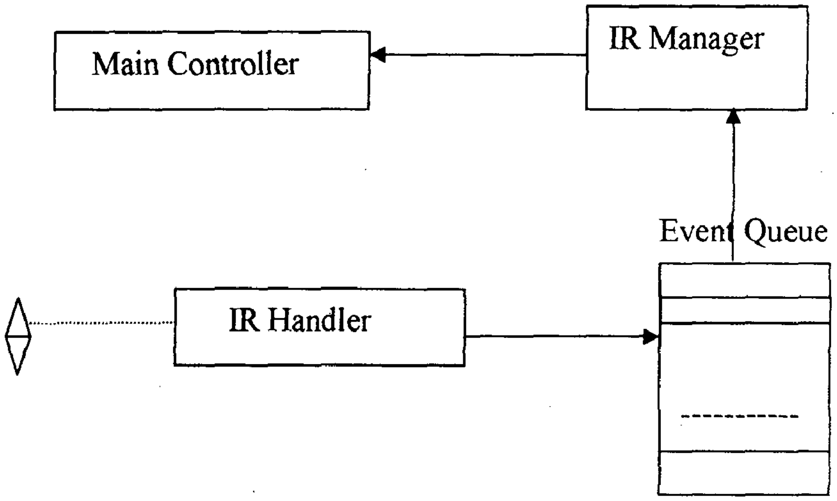


Figure No. 3.1 Various OSD/Set-top Box Logical Layout

The above figure shows the logical layout of various parts for the display of various menu's on the TV screen and of the set-top box.

3.2 Functions of various parts

IR Handler: It handles the events such from an IR source (such as pressing of the key on the remote of STB). It scans the events and put them in the Event Queue.

Event Queue: Basically , it is a stack area where all the events are stored.

IR Manager : Its checks for the new events in the Event Queue. It also checks for their validity i.e. whether the events in the queue are okay or not.

Main Controller: It 's function is to call the event key listener corresponding to the event occurred. It has six sub -controllers shown in the figure no. 3.1

3.3 Various Subcontrollers :

Channel UP/DOWN Sub-controller: It is used to control and respond to up/down arrow keys in the remote control of STB thereby resulting in the change of TV channels. By pressing the UP arrow key , channel number get incremented by one and vice-versa.

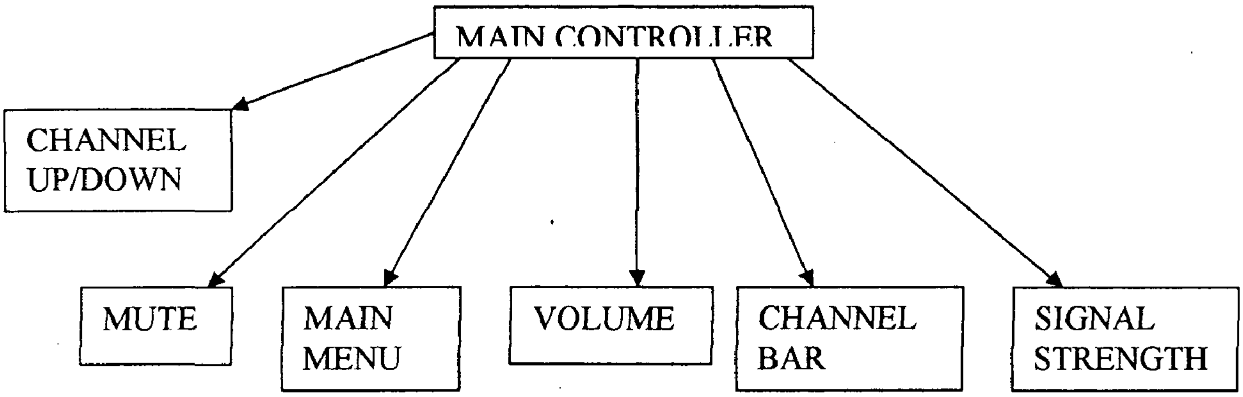


Figure No. 3.2 -Various Sub controllers

Mute subcontroller : It's function is to make a channel soundless when the user presses mute key of remote control of STB.

Main Menu subcontroller : It controls the various options in the menus of OSD of STB. It starts with the showing of Main Menu of the OSD.Refer figure no.3.2

Volume subcontroller : It's job is to increase/decrease the volume of a running channel with a progressive bar which get displayed on the TV screen when the user presses the volume UP/DOWN keys on the remote.

Channel Bar subcontroller : It is used to control and display the digits position for channel number on TV screen.

Signal Strength subcontroller : It displays the strength of received signal with a progressive bar on the screen.

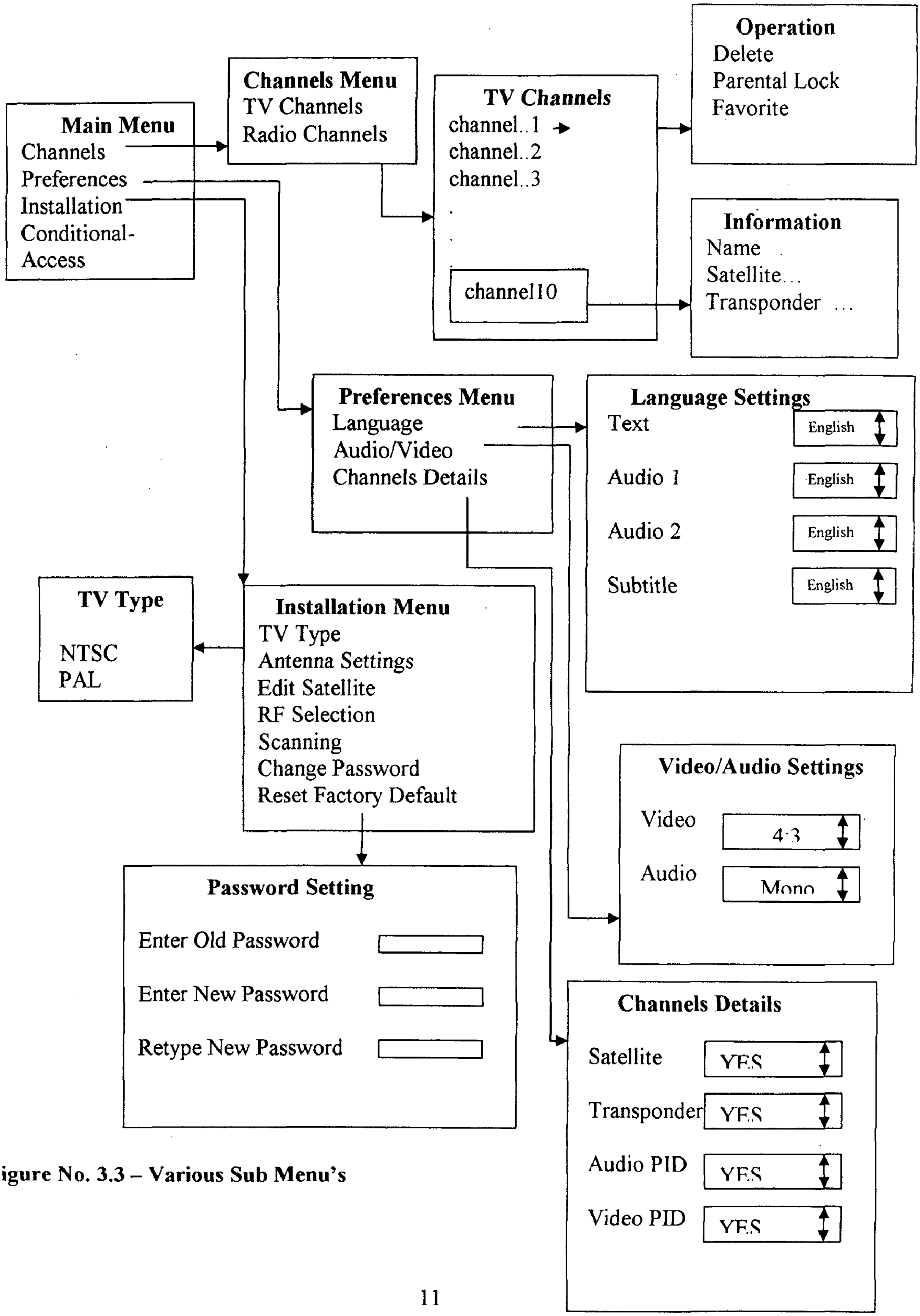


Figure No. 3.3 – Various Sub Menu's

3.4 Various Classes of OSD

3.4.1 Various Class Hierarchies

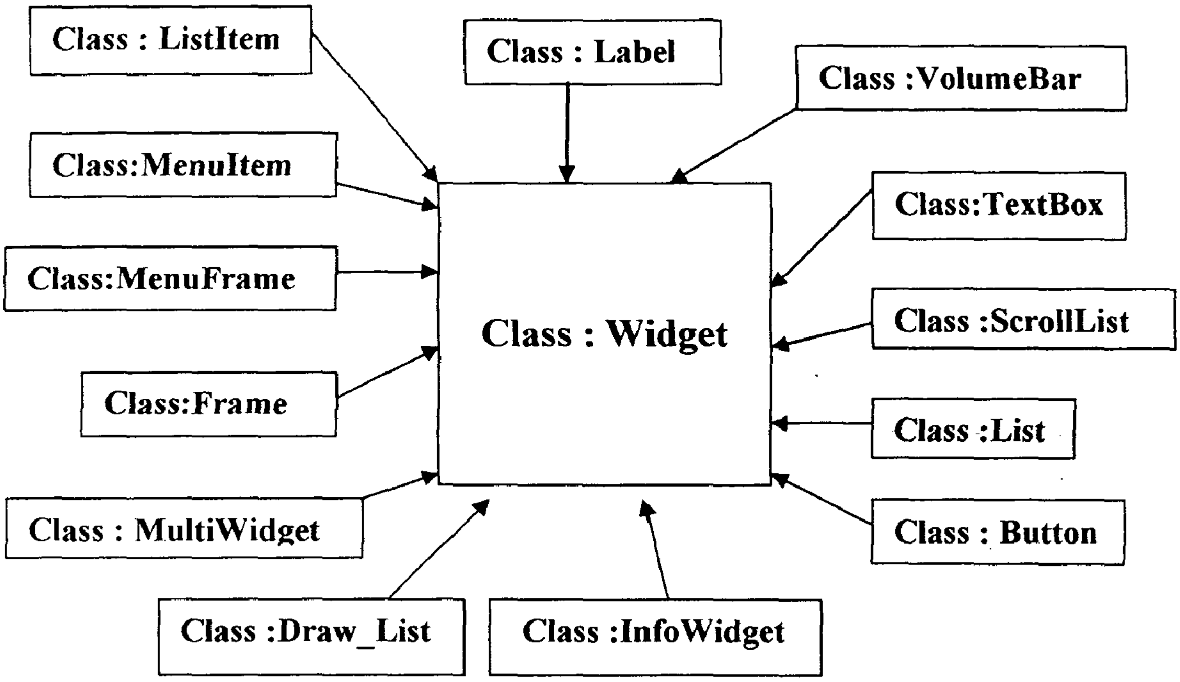


Figure No. 3.4 – Class Hierarchies and Inheritences

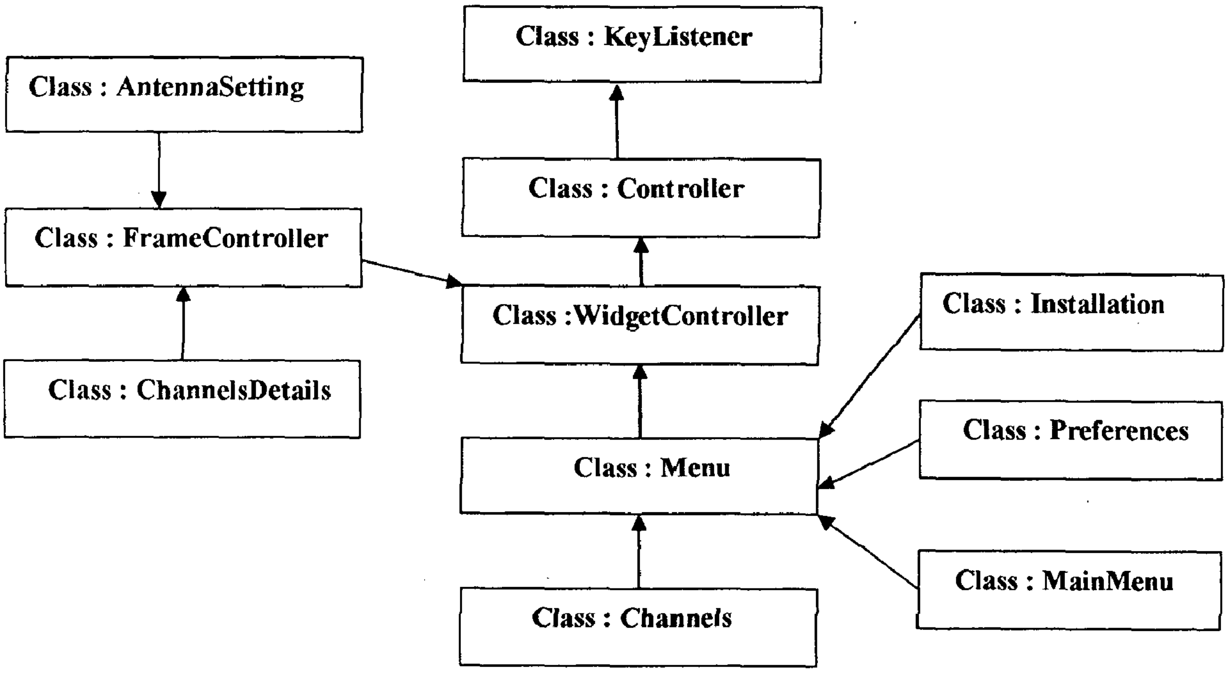


Figure No. 3.5 – Class Hierarchies and Inheritences

3.4.2 Various Classes and their members

Following are the classes and their private and public members :

Class : Label

private members:
unsigned int front_colour_,back_colour;
public members:
virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0; virtual void setFrontColor(us int); virtual void setBackColor(us int);

Class: Widget

protected members:
Widgetcontroller *controller_; Char title_[20];enum mode_;
public members:
unsigned int x_,y_,height_;width_; bool in_focus_;
virtual void paint()=0; Virtual void update()=0; virtual bool setFocus()=0; virtual bool keyEvent (keyEvent ev0=0; virtual void setText(const char*); Virtual void setController(widgetcontroller*controller) { Controller_=controller;}
virtual void setSize (unsigned int x,y,width,height) {x_=x; y_=y; width_=width; height_=height;} widget(const char*title,int x=y=width=height=0);
widget( ){ } ~Widget( ) { }

Class :ListItem

public members:
virtual void paint() =0; virtual void update(); virtual void setFocus()=0;

Class :MenuItem

public members:
virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0;

Class:MenuFrame

public members:
virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0;

Class:Frame

public member:
virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0; Frame() { } Frame (const char*title,intx,y,width, height):Widget(--) { } Frame (int x,y,width,height) :Widget(--) { }

Class :MultiWidget

public members:

virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0;

Class :Draw_List

public members:

virtual void paint() =0; virtual void update(); virtual void keyEvent(KeyEvent)=0;
virtual void activate(Tfunctor * pdf,Mp_OsdDataTYPE*podd);
virtual void add(ListItem*); virtual void add(ListItem *,unsigned int);

protected members:

ListItem *widget[20];

Class : InfoWidget

public members:

virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0; virtual void clear(); virtual void show (const char *);

private members:

InfoWidget(Mp_OsdDataTYPE *); static infoWidget *infowidget_;
Mp_OsdDataTYPE *osdData;

Class :VolumeBar

public members:

virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void clear(); static VolumeBar *getInstance();

private members:

VolumeBar(Mp_OsdDataTYPE *); static VolumeBar *volumebar;
Mp_OsdDataTYPE*osd Data_; bool painting;

Class:TextBox

public members:

virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0; Virtual bool getValue(int *value); virtual void setLength(unsigned int len) { Max_length=len;} virtual char *getText() {data[count]=0;return data;}

protected members:

unsigned char data[20]; unsigned int count,key_pos,max_length_;

Class :ScrollList

public members:

virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0; virtual void add(const char *item); virtual bopol getValue(int *) virtual unsigned int getSeletedItemindex(){ Return selected_item;}

ptotected members:

char *scolllitem_[10]; unsigned int selected_item_; unsigned int no_of_item;

Class : List

public members:
virtual void paint() =0; virtual void update(); virtual void keyEvent(KeyEvent)=0; virtual void setheader(const char*); virtual void add(ListItem *); virtual void add(ListItem *,unsigned int);

Class :Button

public members:
virtual void paint() =0; virtual void update(); virtual void setFocus()=0; virtual void keyEvent(KeyEvent)=0;

Class : FrameController

public members:
P:U;K;A;O;
protected members:
unsigned int x_,y_,width_,height_; Frame frame;

Class : Key Listener

public members:
virtual voidkeyEvent(Key event ev)=0;

Class : Controller

public members:
virtual Mp-OsdataTYPE*getOsd(){return osdData_:} virtual void keyEvent{keyEvent ev}=0; virtual void activate{Tfuctor **pfn, Mp_OsdDataTYPE**podd); Controller(){osdData_=0;p_nfy_=0;}
protected members:
MpOsdData TYPE *osdData_; Tfunctor *pnfy:

Class : WidgetController

public members:
K; // Pure virtual function; A;O;
protected members:
unsigned int no_items_, selecteditemindex_; Widget *widgets_[MAX_COMPONENTS]; Word * prev_osd_data;

Class : Menu

public members:
P;U;K;A;O; virtual void add(Menuitem*); virtual void add(Menuitem *, unsigned int); virtual void setheader(const char*); virtual void setfooteer(const char*);
protected members:
void show(); unsigned int x_., y_, width_, height_; MenuFrame menuframe; Label *topLabel, *bottomLabel;

Class : Channels

public members:

K;A;O; Void channels_callback();

protected members:

unsigned int x_,y_,width_,height_; MenuFrame menuframe; Label *topLabel, *bottomLabel;

Class : Installation

public members:

K;A;O; void installation_callback();

protected members;

unsigned intx_,y_,width_,height_; MenuFrame menuframe; Label *topLabel, *bottomLabel; FrameController *controller[3]; Tfunctor *call_back_fn;

Class :Preferences

public members:

K;A;O;

protected members:

unsigned int x_,y_,,width_,height_; Menuframe menuframe; Label *topLabel, *bottomLabel; FrameController *controller[3]; Tfunctor *call_back_fn;

Class :MainMenu

public members:

K;A;O;

protected members:

unsigned intx_,y_,width_,height_; MenuFrame menuframe; Label *topLabel, *bottomLabel; FrameController *controller[3]; Tfunctor *call_back_fn;

Chapter 4

OSD DATA STRUCTURES AND FUNCTIONS

4.1 Introduction

The OSD-functions provide functions for displaying text and graphics within certain areas of the display-screen called “OSD-region”. The MPEG-decoder allows to display several “OSD-regions” at once, with the restriction that they are positioned in a vertical order, which means that two regions can not share a common horizontal line.

The API-Library supports .up to 8-regions by default. This number can be changed by the constant definition of “MAX_OSDS” in the file “MP_API.h”.

4.2 OSD Data Structure

For passing parameters to OSD-Functions, an approach was used to set-up the relevant parameters in a dedicated data-structure called “OSD data structure” and then only pass a pointer to this data-structure to the OSD-function.

This data structure has the following definition:

```
typedef struct {              // Actual Information for OSD-Graphics Routines
WORD OsdIndex;                // Index of OSD, must be same as nth OSD-region allocated.
                              (0=1st, 1=2nd, etc.)
 BYTE *pSourceData;           // Pointer to BitMap of Source-Data
 BYTE *pShadowData;           // Pointer to BitMap of New/Old Background-Data
 WORD nXBits;                 // specify nHoriz-Bits
 WORD nYLine;                 // specify nVert-Lines
 WORD nSrcLeftSh;             // specify n-Bits Souce-Data Left-shift (to drop 1..7 bits)
 WORD nOfsToNxBt;             // Next xByteOffset for next Vert. Line
 WORD ActXPos;                // Actual X-Pixel Position in OSD
 WORD ActYPos;                // Actual Y-Line Position in OSD
 WORD FrontColour;            // Actual Front Colour index
 WORD BackgColour;            // Actual Background Colour index
 WORD DrawFlags;              // Flags used for some drawing procedures
 WORD OsdXRes;                // number of Horiz. Pixels (X-Resolution)
 WORD OsdYRes;                // number of Vert. Lines (Y-Resolution)
 WORD OsdHSPos;               // OSD Horizontal Start Position on Screen
 WORD OsdVSPos;               // OSD Vertical Start Positin on Screen
 WORD nColours;               // 16 or 256 Colors supported yet
 WORD OsdFlags;               // Bit3..0:AlphaValue, Bit4:HorizDoubling,
Bit5:Vert.Doubling(not supported yet)
 WORD *pColourTable;          // pointer to ColorTable, NULL means no new color table
} Mp_OsdDataTYPE;
```

BYTE – it is equivalent to ‘short int ‘ i.e. it requires 8 bits.

WORD -it is equivalent to ‘int ‘ i.e. it requires 16 bits.

4.3 OSD Functions

Following are the various functions used in the development of the OSD graphical user interface.

void Mp_ResetOsdAlloc();

This function can be used to cancel and reset all current OSD-memory allocation and to switch off any currently active OSD-regions. It is recommended to call this function always before allocating a new set of OSD-regions, in case there were some regions allocated before.

OPTIRET Mp_GetFreeOsdMemWSize();

This function can be used to get the size of available memory in the MPEG-SDRAM for OSD-allocation in WORDs.
Note that the size depends upon if video is currently decoded, if the so called "B-buffer-compression" is activated, and the number and buffer sizes allocated for section-filtering.The function is mainly available for debug purpose, to get a feeling on how big an OSD-region can be opened under above mentioned conditions.

OPTIRET Mp_AllocOSD(Mp_OsdDataTYPE *pODD);

This function is used to allocate the memory for a new OSD-region and also specifies the colour-mode for this region. The following variables in the "OSD data structure" must be specified:

OsdIndex	Index of OSD, must be same as nth OSD-region allocated. (0=1st, 1=2nd, etc.)
OsdXRes	Number of Horizontal Pixels (X-Resolution)
OsdYRes	Number of Vertical Lines (Y-Resolution)
OsdHSPos	Horizontal start-position of OSD-region on display-screen
OsdVSPos	Vertical start-position of OSD-region on display-screen
nColours	number of colours (colour mode), 16 or 256 Colours supported yet
OsdFlags	Bit3..0: Alpha-blending-value for colours with active transparency-bit Bit4: Horizontal Doubling-flag Bit5: Vertical Doubling flag
*pColourTable	Pointer to a Colour Table, NULL means no new colour table required ("Mp_Default256ColourTable" or "Mp_Default16ColourTable" can be used to specify some default colours.)

* Note that the co-ordinate-system for the OSD-function starts in the upper left corner.

The amount of free memory for the OSD-allocation is dependent on the actual state of video-decoding. If no video-decoding is active (video switched off), there is a large amount of memory available for multiple maximum size OSDs, even in 256 color mode.
If video-decoding is active (video channel is running), there is only some limited amount of memory available for OSDs. Thus, in this case, it is not possible to allocate a maximum size OSD.If video-decoding is using the 'compressed-mode', there is some more memory available.

How much memory is available in each of these 3 cases depends also on the number and buffer-sizes for the 'data-stream' allocation.

OPTIRET Mp_SetOsdPosParam
(Mp_OsdDataTYPE *pODD,WORD xSize,WORD ySize, OPTIPAR PosFlag);

The previous function "Mp_AllocOSD()" needs a lot of parameters to be set-up in the "OSD data structure". This can be done either in the data-structure declaration or during program execution by assignment statements. To reduce the number of assignment statements and to avoid having to work out absolute display positions, this function can be used to set-up the absolute position parameters. These are derived from simple size parameters and some location flags, which are passed as arguments. The parameters are:

pODD	Must point to the "OSD data structure" to be set-up.
xSize	Specifies the OSD-region horizontal size in pixel-units.
ySize	Specifies the OSD-region vertical size in line-units.
PosFlag	Specifies some positioning and size-flags.

For PosFlag, the following flags can be specified, also combined by logical OR:

OSD_SET_CENTER	Position the OSD-region in the middle of display
OSD_SET_TOP	Position the OSD-region in the top of the display
OSD_SET_BOTTOM	Position the OSD-region in the bottom of the display
OSD_SET_LEFT	Position the OSD-region at the left side of the display
OSD_SET_RIGHT	Position the OSD-region at the right side of the display
OSD_SET_HDOUB	OSD-region shall be horizontally doubled (Hardware-Feature)
OSD_SET_VDOUB	OSD-region shall be vertically doubled (Hardware-Feature)

After calling this function, the following variables in the "OSD data structure" have been filled-in:

OsdXRes	Number of Horizontal Pixels (X-Resolution) (same as 'xSize')
OsdYRes	Number of Vertical Lines (Y-Resolution) (same as 'ySize')
OsdHSPos	Horizontal start-position of OSD-region on display-screen (depends on 'PosFlag')
OsdVSPos	Vertical start-position of OSD-region on display-screen (depends on 'PosFlag')
OsdFlags	Bit3..0: Alpha-blending-value (remains unchanged) Bit4: Horizontal-doubling-flag (OSD_SET_HDOUB) Bit5: Vertical-doubling flag (OSD_SET_VDOUB)

Thus it might be handy, to call this function before allocating an OSD-region by the function "Mp_AllocOSD()".

OPTIRET Mp_ActivateOsd(OPTIPAR OsdOn, OPTIPAR OsdOff);

This function is used to activate a certain OSD-region. The "OsdOn"-parameter is the OSD-index and specifies which one of the currently available (allocated) OSD-regions should appear actively on the display.
If this OSD-shall replace a different one, for example at the same position with different content (double-buffer concept), use the "OsdOff"-parameter is the index to the OSD which shall be switched off. If no other OSD-region shall be switched off, use the same index value for "OsdOff" as for "OsdOn".

OPTIRET Mp_DeActivateOsd(OPTIPAR OsdN);

Use this function to deactivate a currently visible OSD-region. Which one is specified by the index-parameter "OsdN".

OPTIRET Mp_FillOSD(Mp_OsdDataTYPE *pODD);

This function is used to initialise the display-memory of an OSD-region, thus filling it with a background colour. The following variables in the "OSD data structure" are used in this case:

OsdIndex	Index of OSD, must be same as nth OSD-region allocated. (0=1st, 1=2nd, etc.)
BackgColour	Index of background-colour, same as index in the specified colour-table.

OPTIRET Mp_OsdDrawPixel(Mp_OsdDataTYPE *pODD);

This function can be used to set a pixel into the specified OSD-region. The following variables in the "OSD data structure" are used in this case:

OsdIndex	Index of OSD, must be same as nth OSD-region allocated. (0=1st, 1=2nd, etc.)
ActXPos	defines the X-pixel position in the OSD-region
ActYPos	defines the Y-line position in the OSD-region
FrontColour	Index of pixel-colour, same as index in the specified colour-table.

OPTIRET Mp_OsdDrawRect(Mp_OsdDataTYPE *pODD);

This function can be used to draw rectangular boxes of a certain colour within the OSD-region. The following variables in the "OSD data structure" are used in this case:

OsdIndex	Index of OSD, must be same as nth OSD-region allocated. (0=1st, 1=2nd, etc.)
ActXPos	defines the X-pixel position in the OSD-region
ActYPos	defines the Y-line position in the OSD-region
nXBits	Number of Horizontal Pixels (X-Size)
nYLine	Number of Vertical Lines (Y-Size)
FrontColour	Index of colour, same as index in the specified colour-table.

OPTIRET Mp_MonoDataIntoOSD(Mp_OsdDataTYPE *pODD);

This function can be used for example to transfer character-bitmaps into an OSD-region.It converts monochrome (black/white) bit-map-data into front- or background- colour pixels of the specified colours. If the bit-map data contains a '1'-bit, the associated OSD-pixel is set to the specified front-colour. If the bit-map data contains a '0'-bit, the associated OSD-pixel is set to either the specified background-colour or this pixel remains unchanged keeping its original colour. Which option is taken, depends on the so-called background-bit-map data. The following variables in the "OSD data structure" are used in this case:

OsdIndex	Index of OSD, must be same as n^th^ OSD-region allocated. (0=1st, 1=2nd, etc.)
*pSourceData	pointer to front bit-map data
*pShadowData	pointer to background bit-map-data
nXBits	specifies the number of horizontal pixels to process
nYLine	specifies the number of vertical pixel-lines to process
nSrcLeftSh	the front/background bit-map data can be left-shifted by 1..7 bits, before it is used for the transfer operation. This might be useful if the bit-map data does not start at the MSB position.
NOfsToNxBt	defines the byte-offset for pointing to the next line of bit-map-data
ActXPos	defines the target X-pixel position in the OSD-region where the transfer starts
ActYPos	defines the target Y-line position in the OSD-region where the transfer starts.
FrontColour	Index of front-colour, same as index in the specified colour-table.
BackgColour	Index of background-colour, same as index in the specified colour-table.

4.4 Higher Level OSD Functions

The API-Library also contains some higher-level OSD functions.

OPTIRET Mp_TextOutput(CSTR *pStr, BYTE FontNb, Mp_OsdDataTYPE *pODD);

This function can be used to print string messages into an OSD-region.

'*pStr' points to the string to be printed.
'FontNb' is an integer (currently from 0..3) specifying a specific font.
'*pODD' points to the OSD-data structure.

OPTIRET Mp_CopyDataFromOSD(Mp_OsdDataTYPE *pODD);

This function is the reverse of the previously discussed 'Mp_CopyDataIntoOSD()'-function. It can be used to copy data from the OSD-buffer-memory into the 'OSD-ShadowImage' within the processor memory. The variables in the "OSD data structure" used in this case are the same as before.

OPTIRET Mp_CopyDataIntoOSD (Mp_OsdDataTYPE *pODD);

This function can be used to copy data from the "OSD Shadow-Image" into the OSD-buffer memory. The following variables in the "OSD data structure" are used in this case:

OsdIndex — Index of OSD, must be same as nth OSD-region allocated. (0=1st, 1=2nd, etc.)

pSourceData — Pointer to the address of the "OSD Shadow-Image"in the rocessor memory (Note, this might require (BYTE) casting since '*pSourceData' is usually a BYTE pointer)

nXBits — Specifies the number of WORDs per screen-line

nYLine — Specifies the number of vertical lines to copy

NOfsToNxBt — Defines the word-offset for pointing to the next horizontal image-data-line.
(This value might be equal to 'nXBits' or less, if just a part of the "OSD Shadow-Image" is copied)

ActXPos — Defines a destination x-offset in WORD-units, in case the data transfer does not start fully left side.

ActYPos — Defines a destination Y-line offset, in case the transfer does not begin in the 1st OSD-buffer line.

Chapter 5

ON SCREEN DISPLAY MENU'S

5.1 Various On Screen Display (OSD) Menu's

There are four keys in general which are used for following purpose

✓ : This key is used for **OK** selection.

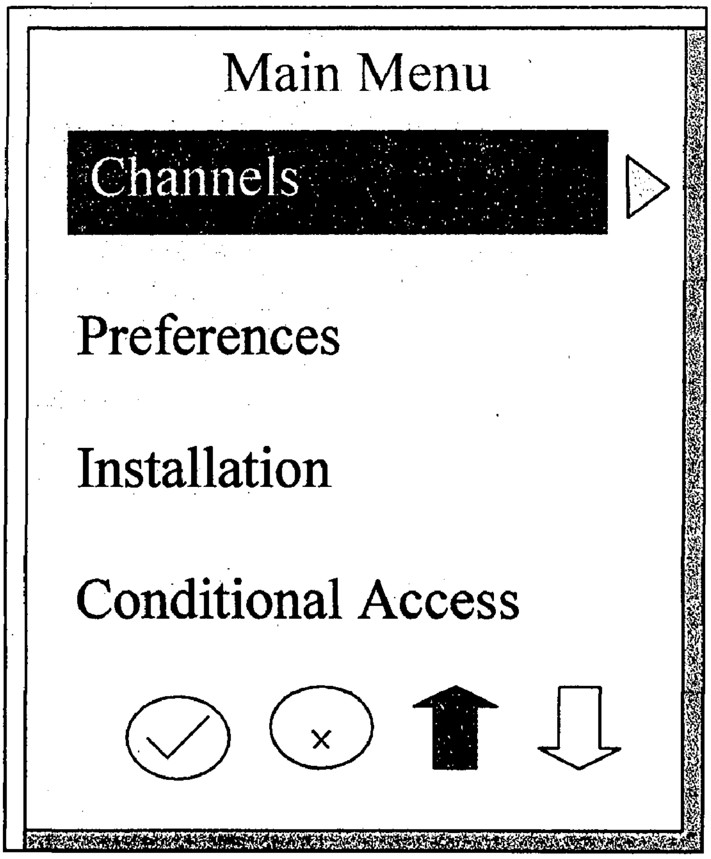
✕ : This key is used for **CANCEL** selection.

⬆ : This key is used for **UP** selection.

⇩ : This key is used for **DOWN** selection.

MAIN MENU:

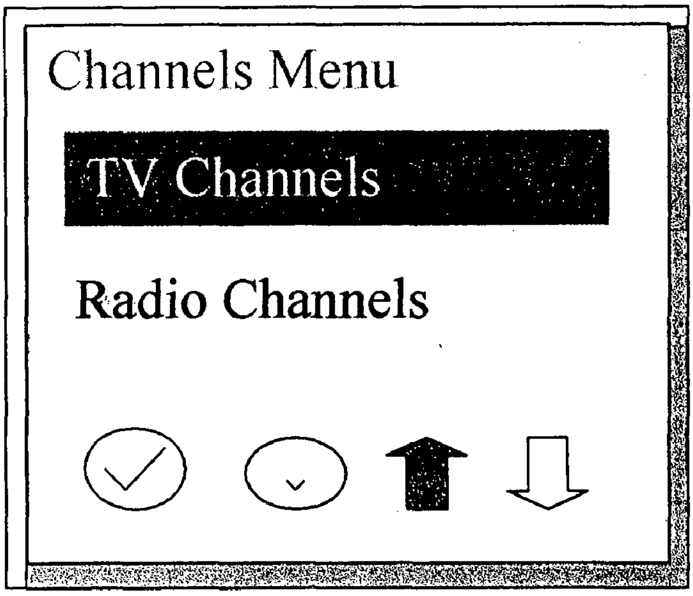
Press ***Menu*** key on remote control ; it will display **Main Menu** Chart



Press ***Exit*** key to come out from Menu

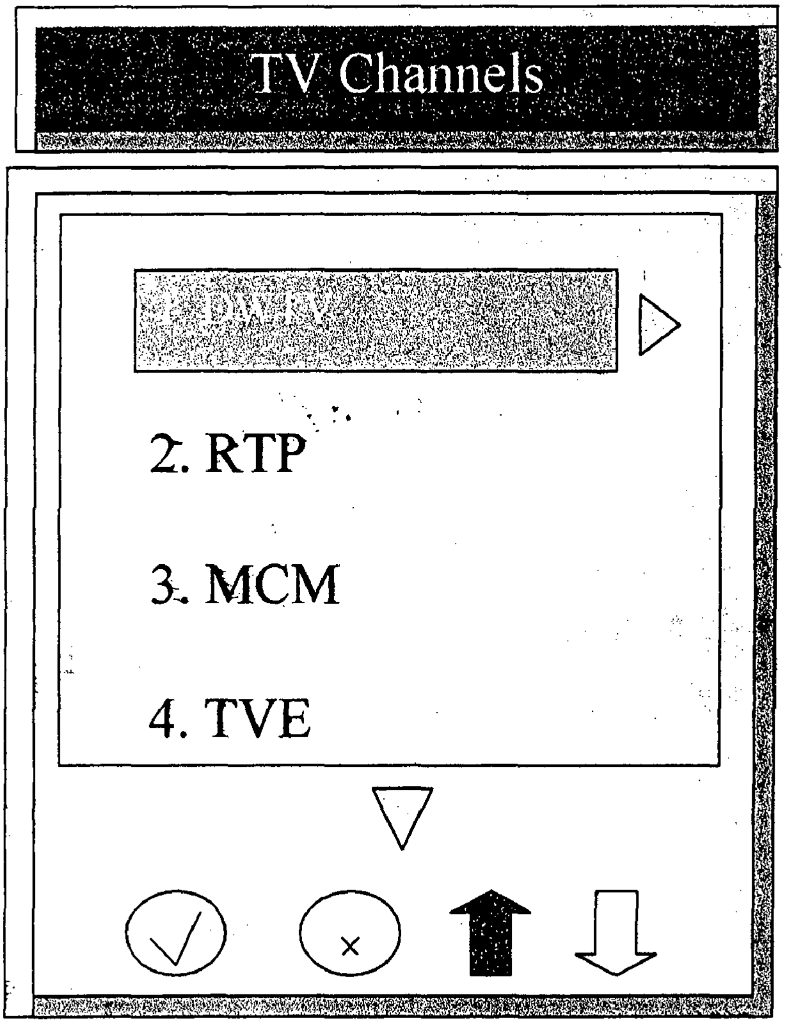
To Select Channel Menu:

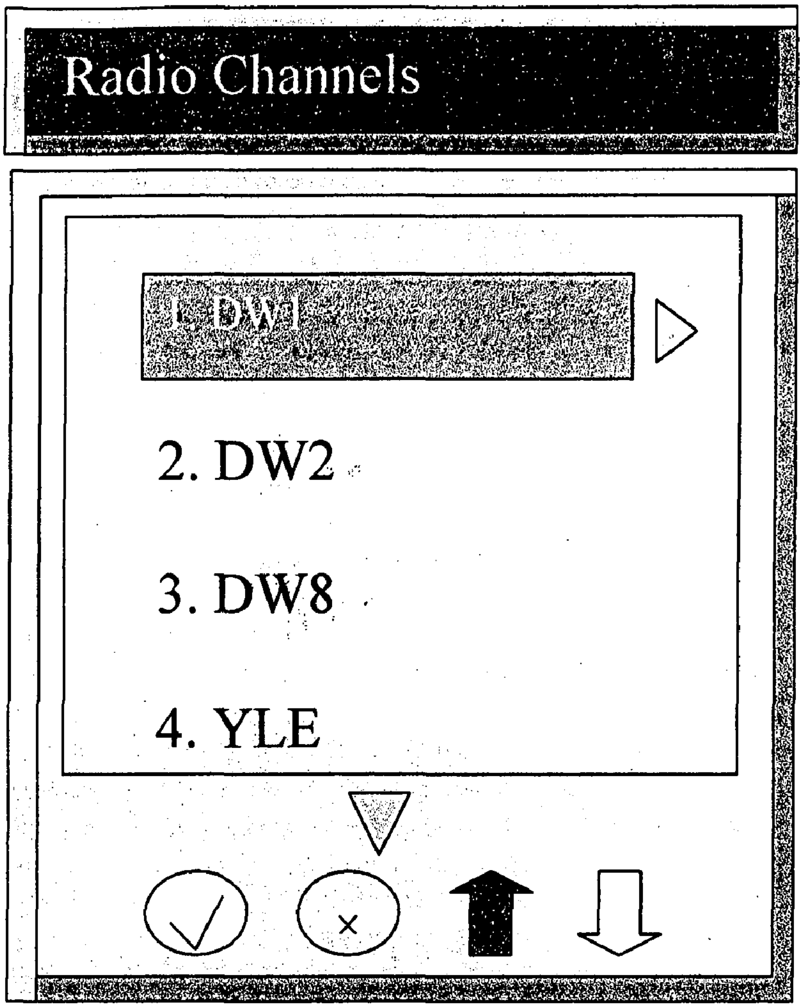
Select **Channels** window from **Main Menu** ; Channel Menu will be displayed



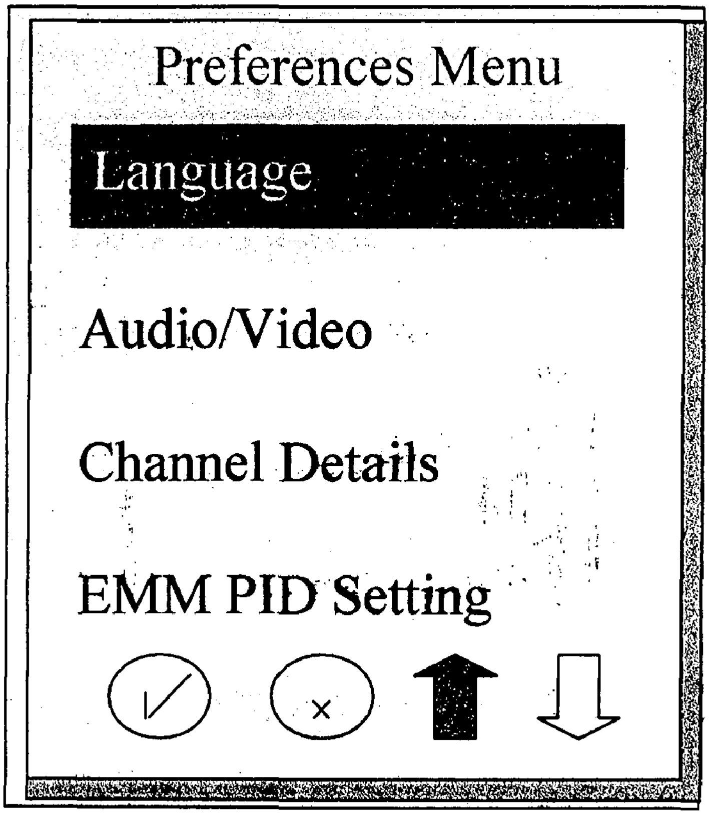
To Select TV/Radio Channels:

Select **TV/Radio Channels** window from **Channel Menu**; list of TV/Radio Channels will be displayed.



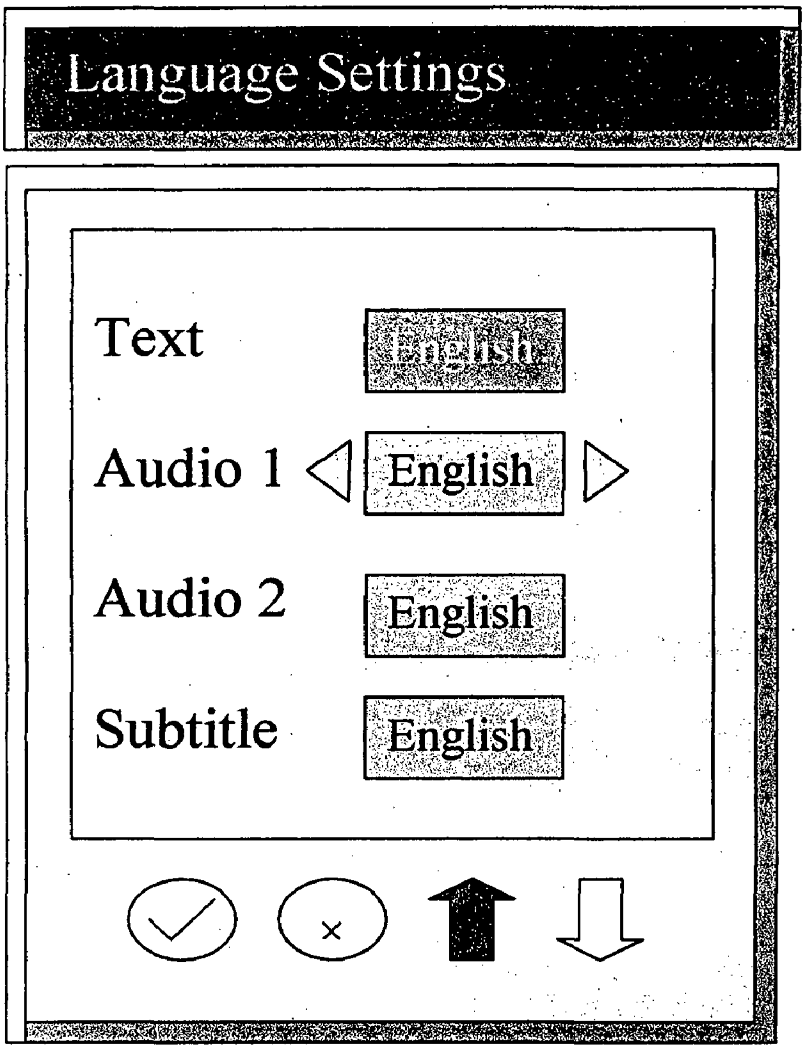


To Select Preferences:
Select **Preferences** window from **Main Menu;** Preferences Menu chart will be displayed



To Select Language:

Select **Language** window from **Preference Menu**; Following will be displayed



Language Settings

Text: English

Audio 1: English

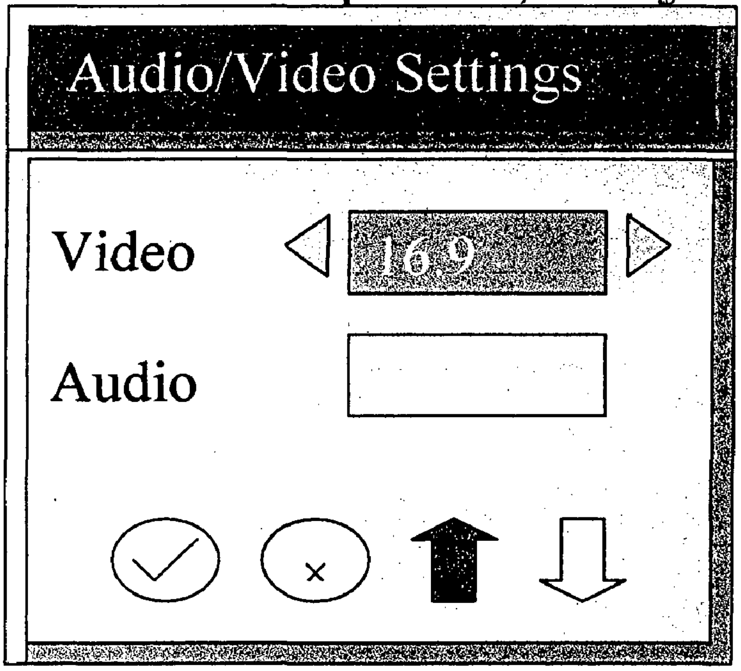
Audio 2: English

Subtitle: English

Press *Left* and *Right* key to select different options of Audio1, Audio2 and Subtitle.

To Select Audio/Video:

Select **Audio/Video** window from **preferences**; following will be displayed



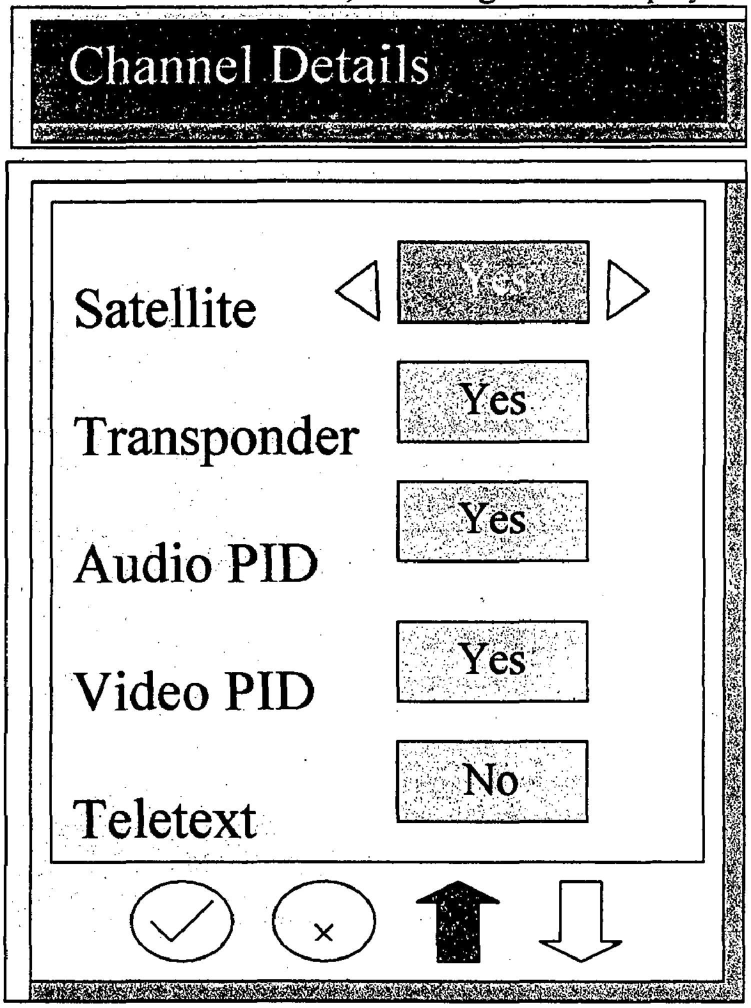
Audio/Video Settings

Video: 16 9

Audio:

To Select Channel Details:

Select **Channel Details** from **Preference Menu**; following will be displayed



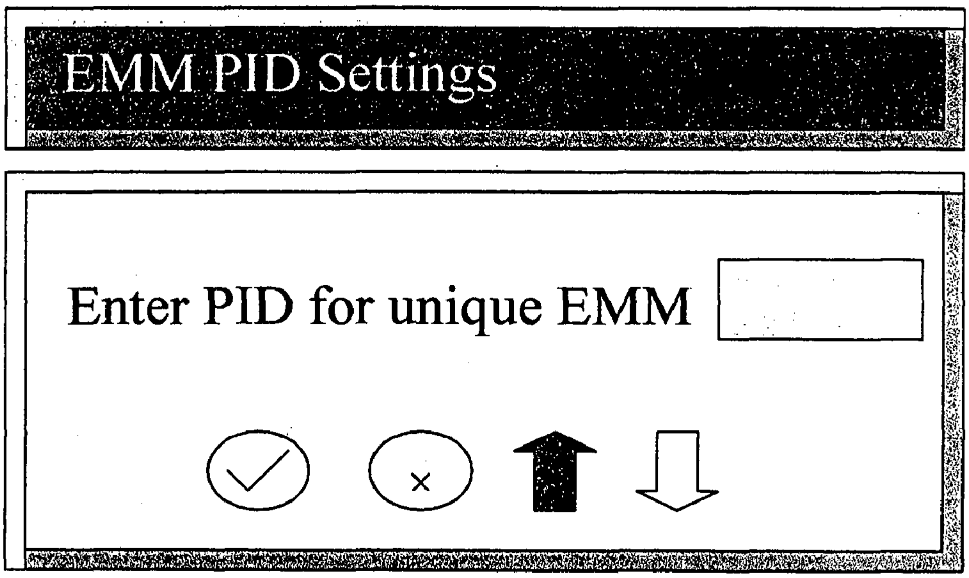
Channel Details

Satellite	◁ Yes ▷
Transponder	Yes
Audio PID	Yes
Video PID	Yes
Teletext	No

Press *Left* and *Right* key to select different options of Satellite, Transponder, Audio PID, Video PID and Teletext.

To Select EMM PID Setting:

Select **EMM PID Setting** from **Preference Menu**.

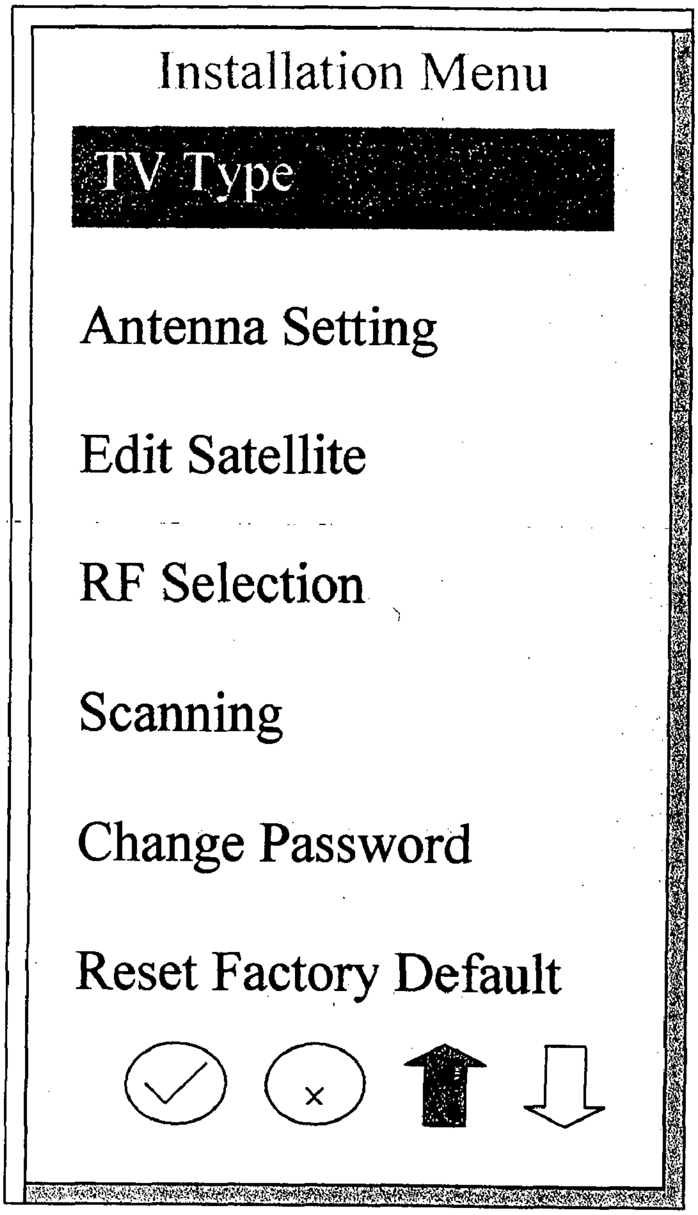


EMM PID Settings

Enter PID for unique EMM □

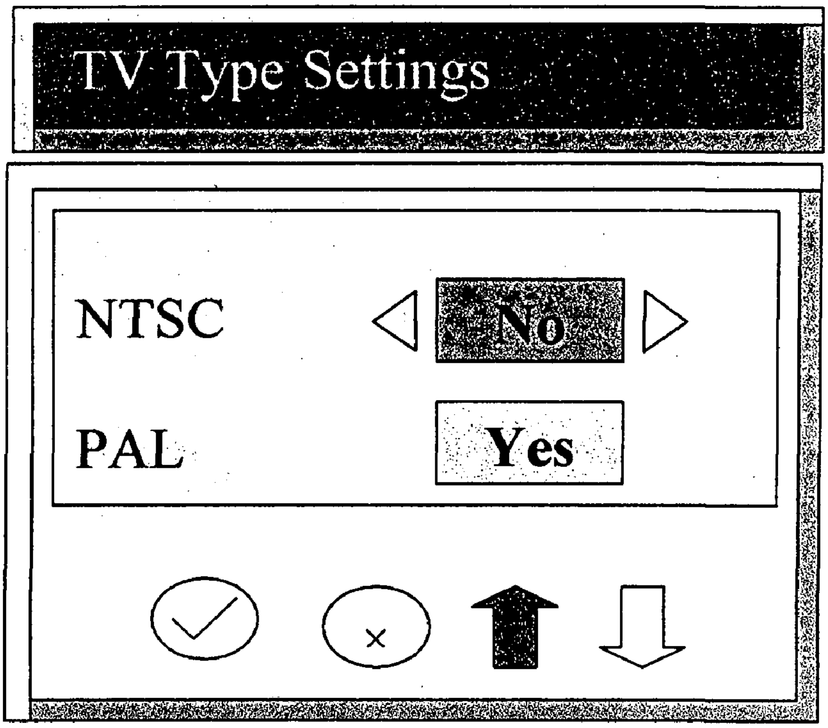
To Select Installation Menu:

Select **Installation** from **Main Menu**; **Installation Menu** Chart will be displayed



TV Type:

Select **TV Type** from **Installation Menu**; following will be displayed



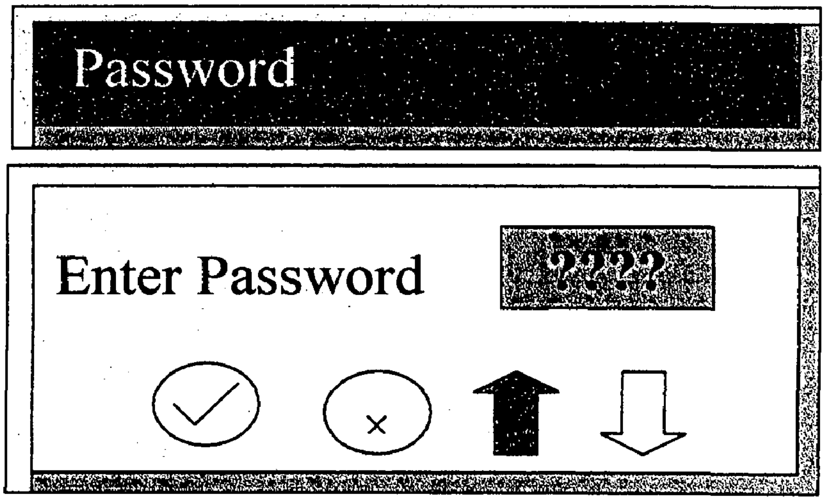
NTSC ◁ No ▷

PAL Yes

Press ***Right*** and ***Left*** key to select different option of NTSC and PAL

Antenna Settings:

Select **Antenna Setting** from **Installation Menu**; following will be displayed



Enter Password ????

Enter password by pressing ***0 to 9*** keys then Press ***OK*** key

TRANSPORT STREAM

6.1 Introduction

The systems part of the **Recommendation/International Standard ISO/IEC ISO/IEC 13818-1: 1994(E) ITU-T Rec. H.222.0 (1995 E)** deals with the combination of one or more elementary streams of video and audio, as well as other data, into single or multiple streams which are suitable for storage or transmission. Systems coding is then carried out as per syntactic and semantic rules imposed by above specification and provides information to enable synchronized decoding of decoder buffers over a wide range of retrieval or receipt conditions.
System coding is specified in two forms:

a. The **Transport Stream** &
b. The **Program Stream**.

Each is optimized for a different set of applications. Both the Transport Stream and Program Stream defined in above standards provide coding syntax which is necessary and sufficient to synchronize the decoding and presentation of the video and audio information, while ensuring that data buffers in the decoders do not overflow or underflow. Both stream definitions are packet-oriented multiplexes.
The video and audio data is encoded and the resulting compressed elementary streams are packetized to produce **PES packets**. The basic multiplexing approach for single video and audio elementary streams is illustrated in figure no. 6.1

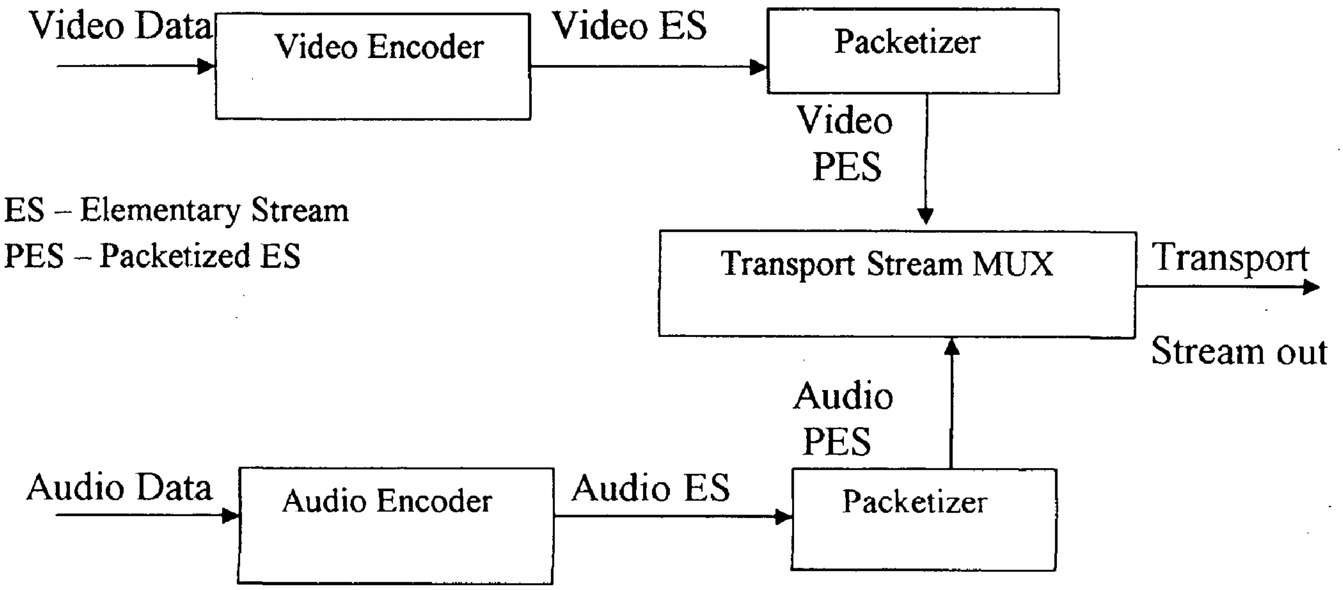


Figure No.6.1 - Transport Stream Encoding

6.2 Transport Stream

The Transport Stream (TS) combines one or more programs with one or more independent time bases into a single stream. PES packets made up of elementary streams that form a program share a common time base. The TS is designed for use in environments where errors are likely, such as storage or transmission in lossy or noisy media. Transport Stream packets are 188 bytes in length.The Transport Stream is a stream definition which is tailored for communicating or storing one or more programs of coded data according to ITU-T Rec. H.262 | ISO/IEC 13818-2 and ISO/IEC 13818-3 and other data in environments in which significant errors may occur. Such errors may be manifested as bit value errors or loss of packets. Transport Streams may be either fixed or variable rate and it is defined by the values and locations of Program Clock Reference (PCR) fields.

The Transport Stream may be constructed by any method that results in a valid stream. It is possible to construct Transport Streams containing one or more programs from elementary coded data streams, from Program Streams, or from other Transport Streams which may themselves contain one or more programs. But in most of the cases, we take a program stream , convert it into a transport stream to carry it over a lossy environment and then recovers a valid and identical program stream.

Transport Streams are constructed in two layers: a system layer and a compression layer. The input stream to the transport stream decoder has a system layer wrapped about a compression layer. Input streams to the Video and Audio decoders have only the compression layer.

6.2.1 Packetized Elementary Stream

Transport Streams is logically constructed from Packetized Elementary Stream (PES) packets. The PES packet format is depicted below :

packet start code prefix	stream id	PES packet length	optional PES HEADER	PES packet data bytes
24 bits	8 bits	16 bits	3 - 256 bytes	upto 65526 bytes

Figure No. 6.2 - PES Packet Format

The semantic definitions of the fields in a PES packet are as follows:

packet_start_code_prefix : It is 24- bit code. Together with the stream_id that follows it constitutes a packet start code that identifies the beginning of the packet. The packet_start_code_prefix is the bit string '0000 0000 0000 0000 0000 0001'(ox000001).

stream_id: It is an 8-bit field. In programs , stream_id specifies the type and number of elementary stream. In TS, stream_id may be set to any valid value which correctly describes the elementary stream.

PES_packet_length : It is 16-bit field specifying the numbers of bytes in the PES packet following the last byte of the field. A value of 0 indicates that the PES packet length is neither specified nor bounded and is allowed only in PES packets whose payload is a video elementary stream contained in TS packets.

Optional PES Header field : It can be 3 to 259 bytes long. It has scrambling control information, priority bits, copyright information , flags, time stamps information , stuffing bytes etc.

PES Packet Data Bytes : It contains the payload and can be up to 65526 bytes long.

PES packets may be much larger than the size of a Transport Stream packet.A continuous sequence of PES packets of one elementary stream with one stream ID may be used to construct a PES Stream. When PES packets are used to form a PES stream, they shall include Elementary Stream Clock Reference (ESCR) fields and Elementary Stream Rate (ES_Rate) fields. The PES stream data shall be contiguous bytes from the elementary stream in their original order and does not contain necessary system information which is there in TS packets.

6.2.2 Timing Model

Systems, Video and Audio all have a timing model in which the end-to-end delay from the signal input to an encoder to the signal output from a decoder is a constant. This delay is the sum of encoding, encoder buffering, multiplexing, communication or storage, de-multiplexing, decoder buffering, decoding, and presentation delays.

As part of this timing model all video pictures and audio samples are presented exactly once and the inter-picture interval and audio sample rate are the same at the decoder as at the encoder. All timing is defined in terms of a common system clock, referred to as a System Time Clock. In the Transport Stream, the system clock frequency is constrained to have the exactly specified ratio to the audio and video sample clocks at all times.

6.2.3 Individual Stream Operations (PES Packet Layer)

The principal stream-specific operations are:

1. De-multiplexing and
2. Synchronizing playback of multiple elementary streams.

De-multiplexing

On encoding, transport streams are formed by multiplexing elementary streams, program streams, or the contents of other TS. Elementary streams may include private, reserved, and padding streams in addition to audio and video streams.

The streams are temporally subdivided into packets, and the packets are serialized. A PES packet contains coded bytes from one and only one elementary stream.

For transport streams, the packet length is 188 bytes. After decoding, de-multiplexing is required to reconstitute elementary streams from the multiplexed Transport Stream with the help of Stream_id codes and Packet ID codes in the transport stream.

Synchronization

Synchronization among multiple elementary streams is accomplished with Presentation Time Stamps (PTS) in the Program Stream and Transport streams. Time stamps are generally in units of 90kHz. Each program in a TS, which may contain multiple programs, may have its own time base. The time bases of different programs within a TS may be different. Because PTSs apply to the decoding of individual elementary streams, they reside in the PES packet layer of both the TS. End-to-end synchronization occurs when encoders save time stamps at capture time, when the time stamps propagate with associated coded data to decoders, and when decoders use those time stamps to schedule presentations.
Synchronization of a decoding system with a channel is achieved through the use of the SCR in the PS and by its analog, the PCR, in the TS. The SCR and PCR are time stamps encoding the timing of the bit stream itself, and are derived from the same time base used for the audio and video PTS values from the same program. Since each program may have its own time base, there are separate PCR fields for each program in a TS containing multiple programs. A program shall have one and only one PCR time base associated with it.

6.2.4 Transport Stream Coding Structure and Parameters

In Transport Stream ,data from each elementary stream are multiplexed together with information that allows synchronized presentation of the elementary streams within a program. A TS consists of one or more programs. Audio and video elementary streams consist of access units. Elementary Stream data is carried in PES packets. A PES packet consists of a PES packet header followed by packet data. PES packets are inserted into Transport Stream packets. The first byte of each PES packet header is located at the first available payload location of a TS packet. The PES packet header begins with a 32-bit start-code that also identifies the stream or stream type to which the packet data belongs. The PES packet header may contain decoding and presentation time stamps (DTS and PTS). The PES packet header also contains other optional fields. The PES packet data field contains a variable number of contiguous bytes from one elementary stream. Transport Stream packets begin with a 4 byte prefix, as shown in the table no. 6.1.

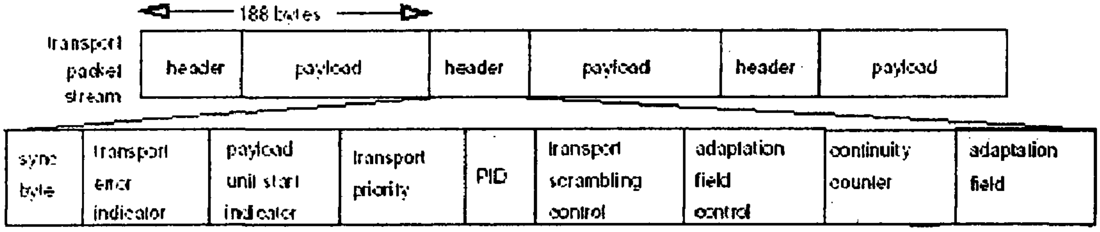


Figure No: 6.3 – TS Packet Format

Syntax	No. of bits	Mnemonics
transport_packet(){		
sync_byte	**8**	**bslbf**
transport_error_indicator	**1**	**bslbf**
payload_unit_start_indicator	**1**	**bslbf**
transport_priority	**1**	**bslbf**
PID	**13**	**uimsbf**
transport_scrambling_control	**2**	**bslbf**
adaptation_field_control	**2**	**bslbf**
continuity_counter	**4**	**uimsbf**
if(adaptation_field_control=='10' || adaptation_field_control=='11'){		
adaptation_field()		
}		
if(adaptation_field_control=='01' || adaptation_field_control=='11') {		
for (i=0;i<N;i++){		
data_byte	**8**	**bslbf**
}		
}		
}		

Table No. 6.1 -- Transport packet

The PID identifies, via the Program Specific Information (PSI) tables, the contents of the data contained in the TS packet. TS packets of one PID value carry data of one and only one elementary stream. The PSI tables are carried in the transport stream. There are four PSI tables namely - Program Association Table (PAT) , Program Map Table (PMT) , Conditional Access Table (CAT) , Network Information Table (NIT). These tables contain the necessary and sufficient information to de multiplex and present programs. TS packets may be null packets. Null packets are intended for padding of transport streams.

6.3 Semantic definition of fields in TS Packet

sync_byte -- The sync_byte is a fixed 8 bit field whose value is '0100 0111' (0x47).

transport_error_indicator -- The transport_error_indicator is a 1 bit flag. When set to '1' it indicates that at least 1 uncorrectable bit error exists in the associated TS packet. This bit may be set to '1' by entities external to the transport layer. When set to '1' ,this bit shall not be reset to '0' unless the bit value(s) in error have been corrected.

payload_unit_start_indicator -- The payload_unit_start_indicator is a 1 bit flag which has normative meaning for Transport Stream packets that carry PES packets or PSI data.When the payload of the Transport Stream packet contains PES packet data, the payload_unit_start_indicator has the following significance:

A '1' indicates that the payload of this Transport Stream packet will commence with the first byte of a PES packet and a '0' indicates no PES packet shall start in this Transport Stream

packet. If the payload_unit_start_indicator is set to '1' then one and only one PES packet starts in this Transport Stream Packet.
When the payload of the Transport Stream packet contains PSI data, the payload_unit_start_indicator has the following significance:
If the Transport Stream packet carries the first byte of a PSI section, the payload_unit_start_indicator value shall be '1', indicating that the first byte of the payload of this Transport Stream packet carries the pointer_field. If the Transport Stream packet does not carry the first byte of a PSI section, the payload_unit_start_indicator value shall be '0', indicating that there is no pointer_field in the payload. For null packets, the payload_unit_start_indicator shall be set to '0'.

transport_priority -- The transport_priority is a 1 bit indicator. When set to '1' ,it indicates that the associated packet is of greater priority than other packets having the same PID which do not have the bit set to '1'. The transport mechanism can use this to prioritize its data within an elementary stream. Depending on the application the transport_priority field may be coded regardless of the PID or within one PID only. This field may be changed by channel specific encoders or decoders.

PID -- The PID is a 13 bit field, indicating the type of the data stored in the packet payload. PID value 0x0000 is reserved for the Program Association Table , PID value 0x0001 is reserved for the Conditional Access Table, PID values 0x0002-0x000F are reserved. PID value 0x1FFF is reserved for null packets. Refer table no. 6.1

transport_scrambling_control -- This 2 bit field indicates the scrambling mode of the Transport Stream packet payload. The TS packet header, and the adaptation field when present, shall not be scrambled. In the case of a null packet, the value of the transport_scrambling_control field shall be set to '00'.

adaptation_field_control -- This 2 bit field indicates whether this TS packet header is followed by an adaptation field and/or payload. Decoders shall discard TS packets with the adaptation_field_control field set to a value of '00'. In the case of a null packet, the value of the adaptation_field_control shall be set to '01'.

continuity_counter -- The continuity_counter is a 4 bit field incrementing with each TS packet with the same PID. The continuity_counter wraps around to 0 after its maximum value. The continuity_counter shall not be incremented when the adaptation_field_control of the packet equals '00' or '10'.

data_byte -- Data bytes shall be contiguous bytes of data from the PES packets, PSI sections, packet stuffing bytes after PSI sections, or private data not in these structures as indicated by the PID. In the case of null packets with PID value 0x1FFF, data_bytes may be assigned any value. The number of data_bytes, N, is specified by 184 minus the number of bytes in the adaptation_field().

Chapter 7

PROGRAM SPECIFIC INFORMATION

7.1 Various PSI Tables

Program Specific Information (PSI) includes both normative data and private data that enable demultiplexing of programs by decoders. Programs are composed of one or more elementary streams, each labeled with a PID. Programs, elementary streams or parts thereof may be scrambled for conditional access. However, Program Specific Information shall not be scrambled. In Transport Streams, Program Specific Information is classified into four table structures as mentioned below. While these structures may be thought of as simple tables, they shall be segmented into sections and inserted in TS packets. The PSI tables are carried in the Transport Stream. There are four PSI tables.

1. Program Association Table (PAT)
2. Program Map Table (PMT)
3. Conditional Access Table (CAT)
4. Network Information Table (NIT)

These tables contain the necessary and sufficient information to demultiplex and present programs. TS packets may be null packets. Null packets are intended for padding of transport streams. A section is a syntactic structure that shall be used for mapping all PSI tables into Transport Stream packets. Along with PSI tables, it is possible to carry private data tables. For this purpose, a private section is defined.

Value	Description
0x0000	Program Association Table
0x0001	Conditional Access Table
0x0002-0x000F	Reserved
0x0010	May be assigned as network_PID,
....	Program_map_PID,elementary_PID,
0x1FFE	or for other purposes.
0x1FFF	Null packet

Table 7.1 -- PID Table

Within a TS, packet stuffing bytes of value 0xFF may be found after the last byte of a section, in which case all following bytes until the end of the TS packet shall also be stuffing bytes of value 0xFF. These bytes may be discarded by a decoder. In such a case, the payload of the next Transport Stream packet with the same PID value shall begin with a pointer_field of value 0x00 indicating that the next section starts immediately thereafter.

Each Transport Stream shall contain one or more Transport Stream packets with PID value 0x0000. These Transport Stream packets together shall contain a complete list of all programs within the Transport Stream. All Transport Stream packets which carry a given TS_program_map_section shall have the same PID value. During the continuous existence of a program, including all of its associated events, the program_map_PID shall not change.

program definition shall not span more than one TS_program_map_section. The maximum number of bytes in a section of a PSI table is 1024 bytes. The maximum number of bytes in a private_section is 4096 bytes. There are no restrictions on the occurrence of start codes, sync bytes or other bit patterns in PSI data.

7.2 Program Association Table

The Program Association Table (PAT) provides the correspondence between a program_number and the PID value of the TS packets which carry the program definition. The program_number is the numeric label associated with a program. The overall table is to be split into one or more *sections with the following syntax. Program number 0x0000 is reserved to specify the network* PID. This identifies the TS packets which carry the Network Information Table.

Each Transport Stream shall contain one or more Transport Stream packets with PID values which are labeled under the program association table as Transport Stream packets containing TS program map sections. Each program listed in the Program Association Table shall be described in a unique TS_program_map_section.

Syntax	No. of bits	Mnemonics
program_association_section() {		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
'0'	**1**	**bslbf**
reserved	**2**	**bslbf**
section_length	**12**	**uimsbf**
transport_stream_id	**16**	**uimsbf**
reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
for (i=0; i<N;i++) {		
program_number	**16**	**uimsbf**
reserved	**3**	**bslbf**
if(program_number == '0') {		
network_PID	**13**	**uimsbf**
}		
else {		
program_map_PID	**13**	**uimsbf**
}		
}		
CRC_32	**32**	**rpchof**
}		

Table No. 7.2- Program Association Section

Any changes in the programs carried within the Transport Stream shall be described in an updated version of the Program Association Table carried in Transport Stream packets with PID value 0x0000.

7.2.1 Table_id Assignments

The table_id field identifies the content of a Transport Stream PSI section as shown in table below:

Value	Description
0x00	Program_association_section
0x01	conditional_access_section(CA_section)
0x02	TS_program_map_section
0x03-0x3F	Reserved
0x40-0xFE	User private
0xFF	Forbidden

Table No. .7.3 - Table-id assignment values

7.2.2 Semantic definition of fields in Program Association Section

table_id -- This is an 8 bit field, which shall be set to 0x00 **section_syntax_indicator** -- The section_syntax_indicator is a 1 bit field which shall be set to '1'.

section_length -- This is a twelve bit field, the first two bits of which shall be '00'. It specifies the number of bytes of the section, starting immediately following the section_length field, and including the CRC. The value in this field shall not exceed 1021

transport_stream_id -- This is a 16 bit field which serves as a label to identify this Transport Stream from any other multiplex within a network. Its value is defined by the user.

version_number -- This 5 bit field is the version number of the whole Program Association Table. The version number shall be incremented by 1 whenever the definition of the Program Association Table changes. Upon reaching the value 31, it wraps around to 0. When the current_next_indicator is set to '1', then the version_number shall be that of the currently applicable Program Association Table. When the current_next_indicator is set to '0', then the version_number shall be that of the next applicable Program Association Table.

current_next_indicator -- A 1 bit indicator, which when set to '1' indicates that the Program Association Table sent is currently applicable. When the bit is set to '0', it indicates that the table sent is not yet applicable and shall be the next table to become valid.

section_number -- This 8 bit field gives the number of this section. The section_number of the first section in the Program Association Table shall be 0x00. It shall be incremented by 1 with each additional section in the Program Association Table.

last_section_number -- This 8 bit field specifies the number of the last section (that is, the section with the highest section_number) of the complete Program Association Table.

program_number -- Program_number is a 16 bit field. It specifies the program to which the program_map_PID is applicable. If this is set to 0x0000 then the following PID reference shall be the network PID. For all other cases the value of this field is user defined. This field shall not take any single value more than once within one version of the program association table. The program_number may be used as a designation for a broadcast channel, for example.

network_PID -- network_PID is a 13 bit field specifying the PID of the Transport Stream packets which shall contain the Network Information Table. The value of the network_PID field is defined by the user, but shall only take values as specified in Table 7.2. The presence of the network_PID is optional.

program_map_PID -- program_map_PID is a 13 bit field specifying the PID of the Transport Stream packets which shall contain the program_map_section applicable for the program as specified by the program_number. No program_number shall have more than one program_map_PID assignment. The value of the program_map_PID is defined by the user, but shall only take values as specified in Table 7.2

CRC_32 -- This is a 32 bit field that contains the CRC value

7.3 Program Map Table

The Program Map Table (PMT) provides the mappings between program numbers and the program elements that comprise them. A single instance of such a mapping is referred to as a "program definition.".The PMT is the complete collection of all program definitions for a TS. This table shall be transmitted in packets, the PID values of which are selected by the encoder. More than one PID value may be used, if desired. The table may be segmented into one or more sections, before insertion into TS packets, with the following syntax. In each section the section number field shall be set to zero. Sections are identified by the program_number field.

Syntax	No. of bits	Mnemonics
TS_program_map_section() {		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
'0'	1	**bslbf**
reserved	**2**	**bslbf**
section_length	**12**	**uimsbf**
program_number	**16**	**uimsbf**
reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
reserved	**3**	**bslbf**
PCR_PID	**13**	**uimsbf**
reserved	**4**	**bslbf**
program_info_length	**12**	**uimsbf**
for (i=0; i<N; i++) {		
descriptor()		
}		
for (i=0;i<N1;i++) {		
stream_type	**8**	**uimsbf**
reserved	**3**	**bslbf**
elementary_PID	**13**	**uimsnf**
reserved	**4**	**bslbf**
ES_info_length	**12**	**uimsbf**
for (i=0; i<N2; i++) {		
descriptor()		
}		
}		
CRC_32	**32**	**rpchof**
}		

Table No. 7.4 – Transport stream Program Map Section

Every program shall be fully defined within the Transport Stream itself. Private data which has an associated elementary_PID field in the appropriate Program Map Table section is part of the program. Other private data may exist in the Transport Stream without being listed in the Program Map Table section. The most recently transmitted version of the TS_program_map_section with the current_next_indicator set to a value of '1' shall always apply to the current data within the Transport Stream. Any changes in the definition of any of the programs carried within the Transport Stream shall be described in an updated version of the corresponding section of the program map table carried in Transport Stream packets with the PID value identified as the program_map_PID for that specific program. Sections with a table_id value of 0x02 shall contain Program Map Table information. Such sections may be carried in Transport Stream packets with different PID values.

7.3.1 Semantic definition of fields in Program Map Section

program_number -- program_number is a 16 bit field. It specifies the program to which the program_map_PID is applicable. One program definition shall be carried within only one TS_program_map_section. This implies that a program definition is never longer than 1016 bytes.

PCR_PID -- This is a 13 bit field indicating the PID of the TS packets which shall contain the PCR fields valid for the program specified by program_number. If no PCR is associated with a program definition for private streams then this field shall take the value of 0x1FFF.

program_info_length -- This is a 12 bit field, the first two bits of which shall be '00'. It specifies the number of bytes of the descriptors immediately following the program_info_length field.

stream_type -- This is an 8 bit field specifying the type of program element carried within the packets with the PID whose value is specified by the elementary_PID.

elementary_PID -- This is a 13 bit field specifying the PID of the Transport Stream packets which carry the associated program element.

ES_info_length -- This is a 12 bit field, the first two bits of which shall be '00'. It specifies the number of bytes of the descriptors of the associated program element immediately following the ES_info_length field.

7.4 Conditional Access Table

The Conditional Access Table(CAT)provides the association between one or more CA systems, their EMM streams and any special parameters associated with them. Whenever one or more elementary streams within a Transport Stream are scrambled, Transport Stream packets with a PID value 0x0001 shall be transmitted containing CA_sections containing CA_descriptors appropriate to the scrambled streams. The transmitted Transport Stream packets shall together form one complete version of the conditional access table. The most recently transmitted version of the table with the current_next_indicator set to a value of '1' shall always apply to the current data in the Transport Stream. Any changes in scrambling making the existing table invalid or incomplete shall be described in an updated version of the conditional access table. These sections shall all use table_id value 0x01. Only sections with this table_id value are permitted within Transport Stream packets with a PID value of 0x0001. The table may be segmented into one or more sections, before insertion into TS packets, with the following syntax:

Syntax	No. of bits	Mnemonics
CA_section() {		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
'0'	**1**	**bslbf**
reserved	**2**	**bslbf**
section_length	**12**	**uimsbf**
reserved	**18**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
for (i=0; i<N;i++) {		
descriptor()		
}		
CRC_32	**32**	**rpchof**
}		

Table No. 7.5 –Conditional Access Section

*** It's fields has been previously defined. Refer PAT.**

7.5 Network Information Table

The Network Information Table (NIT) is optional and its contents are private. If present it is carried within Transport Stream packets that shall have the same PID value, called the network_PID. The network_PID value is defined by the user and, when present, shall be found in the Program Association Table under the reserved program_number 0x0000. If the network information table exists, it shall take the form of one or more private_sections.

7.5.1 Private Section

The use of the private_section is mandatory when private data is sent in TS packets with a PID value designated as a PMT PID in the PAT. This private_section allows data to be transmitted with the absolute minimum of structure while enabling a decoder to parse the stream.
The sections may be used in two ways:

If the section_syntax_indicator is set to '1', then the whole structure common to all tables shall be used; if the indicator is set to '0', then only the fields 'table_id' through 'private_section_length' shall follow the common structure syntax and semantics and the rest of the private_section may take any form the user determines. A private table may be made of several private_sections, all with the same table_id.

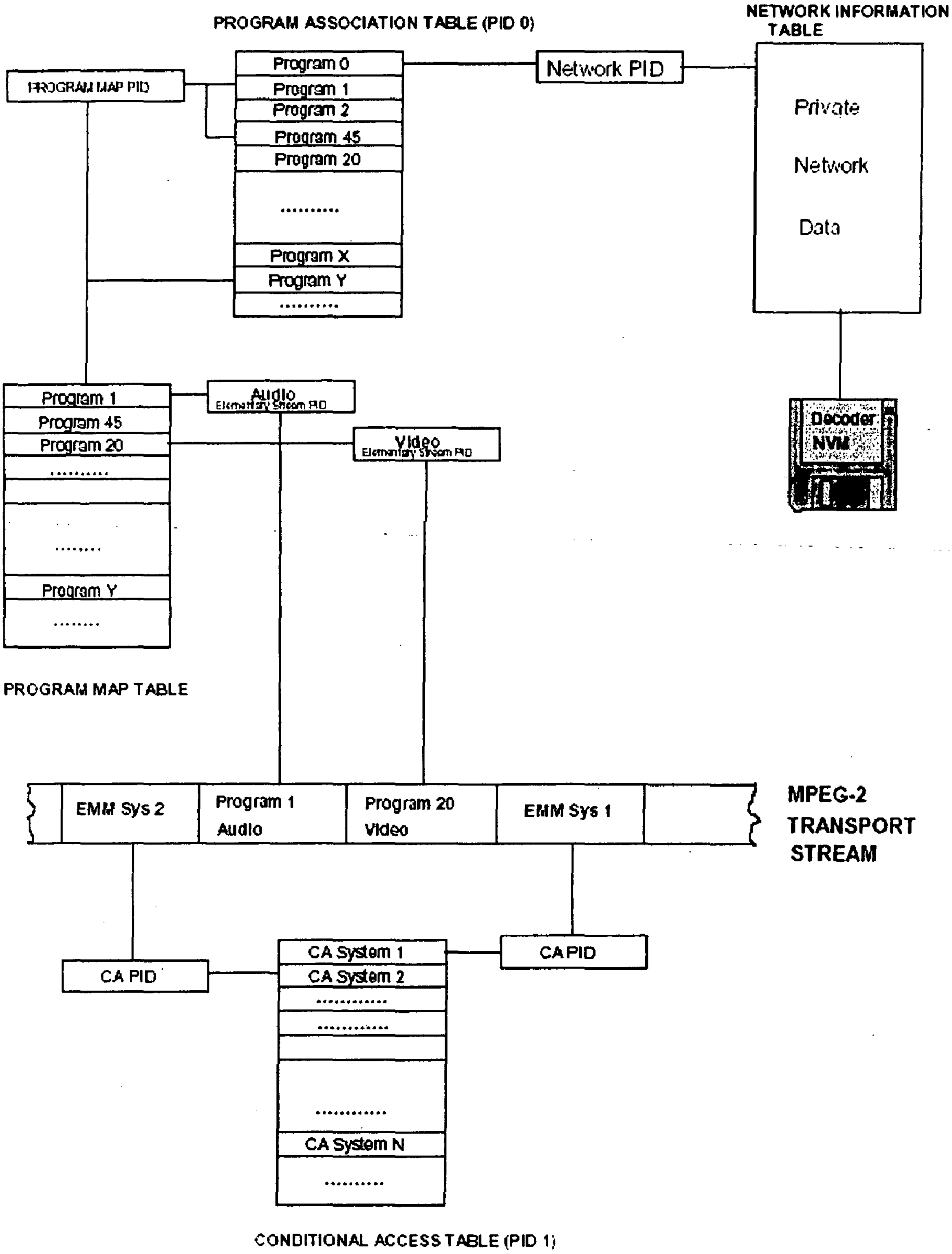


Figure No. 7.1 --Program and Network Mapping Relationships

7.6 Program Element Descriptors

Program element descriptors are structures which have a format which begins with an 8 bit tag value which is followed by an 8 bit descriptor length and data fields.

7.6.1 Semantic definition of fields in Program Element Descriptors

descriptor_tag -- The descriptor_tag is an 8 bit field which identifies each descriptor. Its meaning is given in table 2-40. An 'X' in the TS or PS columns indicates the applicability of the descriptor to either the Transport Stream or Program Stream respectively.

Descriptor_tag	TS	PS	Identification
2	X	X	video_stream_descriptor
3	X	X	audio_stream_descriptor
8	X	X	video_window_descriptor
9	X	X	CA_descriptor

Table No. 7.6 -- Program Element Descriptors

descriptor_length -- The descriptor_length is an 8 bit field specifying the number of bytes of the descriptor immediately following descriptor_length field.

7.7 Video Stream Descriptor

The Video Stream Descriptor (VSD) provides basic information which identifies the coding parameters of a video elementary stream.

Syntax	No. of bits	Mnemonics
video_stream_descriptor(){		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
multiple_frame_rate_flag	**1**	**bslbf**
frame_rate_code	**4**	**uimsbf**
MPEG_1_only_flag	**1**	**bslbf**
constrained_parameter_flag	**1**	**bslbf**
still_picture_flag	**1**	**bslbf**
if (MPEG_1_only_flag == 1){		
profile_and_level_indication	**8**	**uimsbf**
chroma_format	**2**	**uimsbf**
frame_rate_extension_flag	**1**	**bslbf**
reserved	**5**	**bslbf**
}		
}		

Table No. 7.7 -- Video Stream Descriptor

7.7.1 Semantic definitions of fields in Video Stream Descriptor

multiple_frame_rate_flag -- This is a 1 bit field which when set to '1' indicates that multiple frame rates may be present in the video stream. When set to a value of '0' only a single frame rate is present.
frame_rate_code -- This is a 4 bit field except that when the multiple_frame_rate_flag is set to a value of '1' the indication of a particular frame rate also permits certain other frame rates to be present in the video stream.
MPEG_1_only_flag -- This is a 1 bit field which when set to '1' indicates that the video stream contains only data.
constrained_parameter_flag -- This is a 1 bit field which when set to '1' indicates that the video stream shall not contain unconstrained video data.
still_picture_flag -- This is a 1 bit field, which when set to '1' indicates that the video stream contains only still pictures. If the bit is set to '0' then the video stream may contain either moving or still picture data.
profile_and_level_indication -- This is an 8 bit field which is set to the same value as the profile_and_level_indication fields in the video stream.
chroma_format -- This is a 2 bit field. which is set to the same value as the chroma_format fields in the ITU-T Rec. H.262 | ISO/IEC 13818-2 video stream.
frame_rate_extension_flag -- This is a 1 bit flag which when set to '1' indicates that either or both of the frame_rate_extension_n and frame_rate_extension_d fields in the video stream are non-zero.

7.8 Audio Stream Descriptor

The Audio Stream Descriptor (ASD) provides basic information which identifies the coding version of an audio elementary stream.

Syntax	No. of bits	Mnemonics
audio_stream_descriptor(){		
descriptor_tag	8	**uimsbf**
descriptor_length	8	**uimsbf**
free_format_flag	1	**bslbf**
ID	1	**bslbf**
layer	2	**bslbf**
variable_rate_audio_indicator	1	**bslbf**
reserved	3	**bslbf**
}		

Table No. 7.8 -- Audio Stream Descriptor

7.8.1 Semantic definition of fields in Audio Stream Descriptor

free_format_flag -- This is a 1 bit field which when set to '1' indicates that the audio stream data has the bitrate_index set to '0000'. If set to '0' then the bitrate_index is not '0000'.
ID -- This is a 1 bit field which is set to the same value as the ID fields in the audio stream.
layer -- This is a 2 bit field which is set to the same value as the layer fields in the audio stream.
variable_rate_audio_indicator -- This is a 1 bit flag, which when set to '1' indicates that the associated audio stream may contain audio frames in which the bit rate changes. In all cases the audio stream is intended to be presented without any decoding discontinuity

7.9 Video Window Descriptor

The Video Window Descriptor (VWD) is used to describe the window characteristics of the associated video elementary stream.

Syntax	**No. of bits**	***Mnemonics***
video_window_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
horizontal_offset	**14**	**uimsbf**
vertical_offset	**14**	**uimsbf**
window_priority	**4**	**uimsbf**
}		

Table No. 7.9 -- Video Window Descriptor

7.9.1 Semantic definition of fields in Video Window Descriptor

horizontal_offset -- The value indicates the horizontal position of the top left pixel of the current video display window or display rectangle if indicated in the picture display extension on the target background grid for display as defined in the target_background_grid_descriptor. The top left pixel of the video window shall be one of the pixels of the target background grid.

vertical_offset -- The value indicates the vertical position of the top left pixel of the current video display window or display rectangle if indicated in the picture display extension on the target background grid for display as defined in the target_background_grid_descriptor. The top left pixel of the video window shall be one of the pixels of the target background grid.

window_priority -- The value indicates how windows overlap. A value of 0 being lowest priority and a value of 15 is the highest priority, i.e. windows with priority 15 are always visible.

7.10 Conditional Access Descriptor

The Conditional Access Descriptor (CAD) is used to specify both system-wide conditional access management information such as EMMs and elementary stream-specific infor mation such as ECMs. It may be used in both the TS_program_map_section and the program_stream_map. If any elementary stream is scrambled, a CA descriptor shall be present for the program containing that elementary stream. If any system-wide conditional access management information exists within a Transport Stream, a CA descriptor shall be present in the conditional access table.

When the CA descriptor is found in the TS_program_map_section (table_id = 0x02), the CA_PID points to packets containing program related access control information, such as ECMs. Its presence as program information indicates applicability to the entire program. In the same case, its presence as extended ES information indicates applicability to the associated program element. Provision is also made for private data.

When the CA descriptor is found in the CA_section (table_id = 0x01), the CA_PID points to packets containing system-wide and/or access control management information, such as EMMs. The contents of the Transport Stream packets containing conditional access information are privately defined.

7.10.1 Semantic definition of fields in Conditional Access Descriptor

CA_system_ID -- This is an 16 bit field indicating the type of CA system applicable for either the associated ECM and/or EMM streams. The coding of this is privately defined.

CA_PID -- This is an 13 bit field indicating the PID of the Transport Stream packets which shall contain either ECM or EMM information for the CA systems as specified with the associated CA_system_ID. The contents (ECM or EMM) of the packets indicated by the CA_PID is determined from the context in which the CA_PID is found, i.e. a TS_program_map_section or the CA table in the Transport Stream, or the stream_id field in the Program Stream.

Syntax	**No. of bits**	**Mnemonics**
CA_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
CA_system_ID	**16**	**uimsbf**
reserved	**3**	**bslbf**
CA_PID	**13**	**uimsbf**
for (i=0; i<N; i++) {		
private_data_byte	**8**	**uimsbf**
}		
}		

Table No. 7.10 Conditional Access Descriptor

Chapter 8

FLOW CHARTS AND FUNCTIONS

8.1 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bit-stream.

bslbf	Bit string, left bit first
ch	Channel.
rpchof	Remainder polynomial coefficients, highest order first.
tcimsbf	Two's complement integer, msb (sign) bit first.
uimsbf	Unsigned integer, most significant bit first.
vlclbf	Variable length code, left bit first

8.2 Method of describing bit stream syntax

Each data item in the bit stream is described by its name, its length in bits, and a mnemonic for its type and order of transmission. The action caused by a decoded data element in a bit stream depends on the value of that data element and on data elements previously decoded. The following constructs are used to express the conditions when data elements are present.

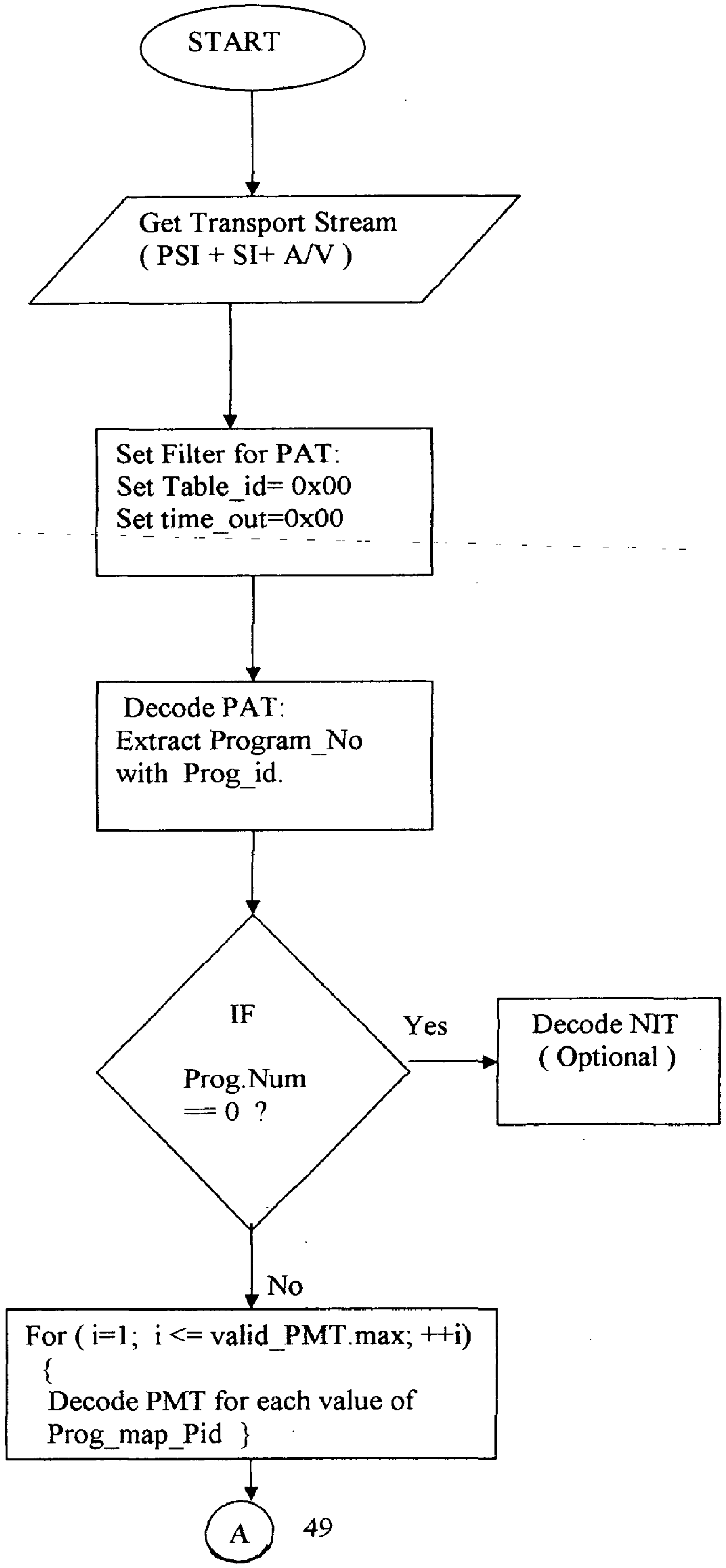
```
while ( condition ) {
data_element
. . .
}  // If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.
```

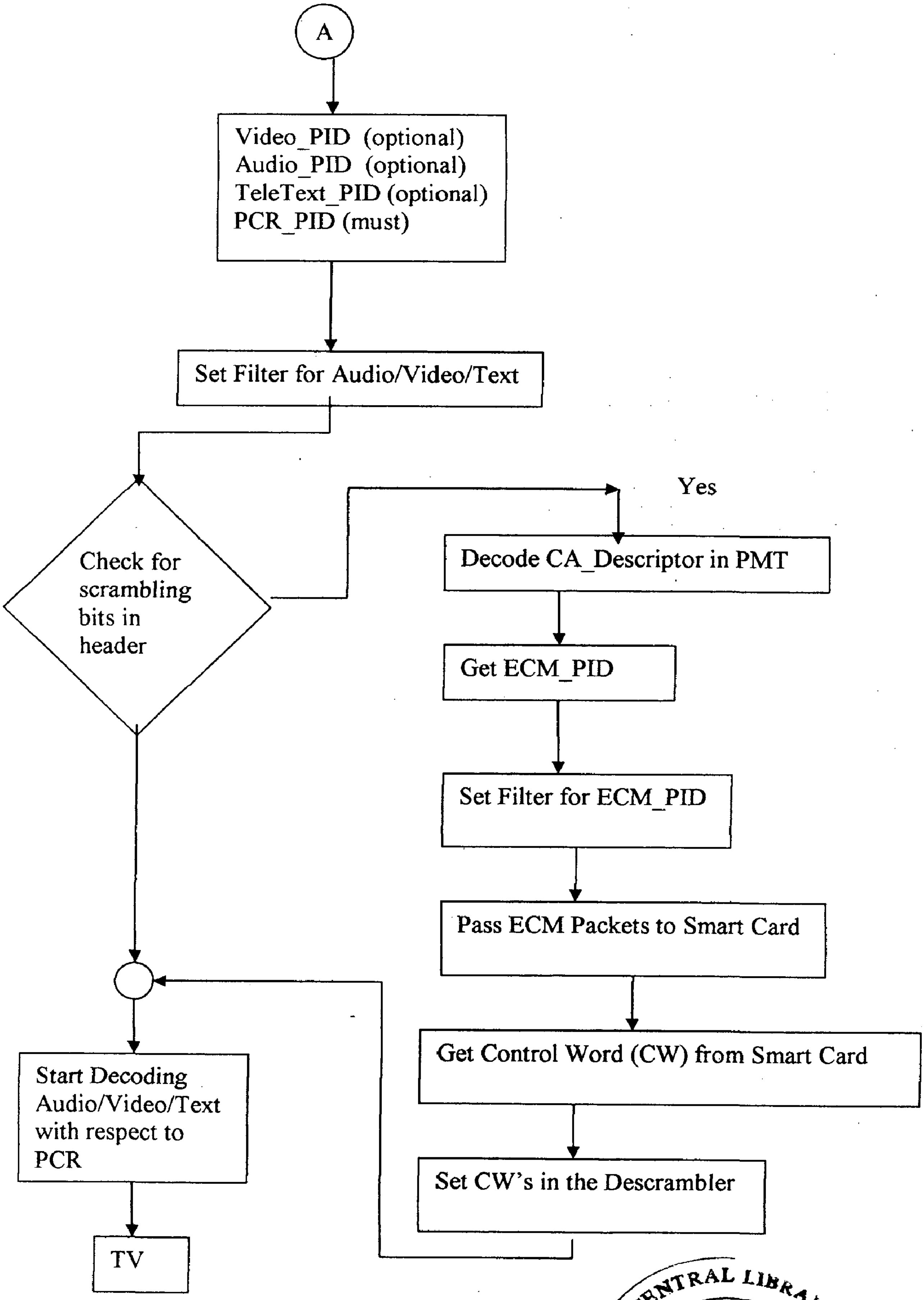
```
do {
data_element
. . . }
while ( condition )  // The data element always occurs at least once. The data element is repeated until the condition is not true.
```

```
if ( condition) {
data_element
. . .
} //If the condition is true, then the first group of data elements occurs next in the data stream.
else {
data_element
. . .
} // If the condition is not true, then the second group of data elements occurs next in the data stream.
for (i = 0;i<n;i++) {
data_element
. . .
}
```

The group of data elements occurs n times depending upon the value of the loop variable i.

8.3 Flowchart for extracting packets from TS




A
Video_PID (optional)
Audio_PID (optional)
TeleText_PID (optional)
PCR_PID (must)
Set Filter for Audio/Video/Text
Check for scrambling bits in header
Yes
Decode CA_Descriptor in PMT
Get ECM_PID
Set Filter for ECM_PID
Pass ECM Packets to Smart Card
Get Control Word (CW) from Smart Card
Set CW's in the Descrambler
Start Decoding Audio/Video/Text with respect to PCR
TV

8.4 Functions to extract information from TS

Following functions have been developed to extract and decode packets from TS:

OPTIRET TD_ProcessDescriptor ():

This function is used for processing of CA_Descriptor in PMT. It takes program index. PMT_PID and total descriptor length as arguments. Read each CA_Descriptor, compare the CA_SYS_ID with the CONAX one and if found matching thenonly process that descriptor further. If match found then copy that descriptor to SI_CAT_INFO structure.

OPTIRET TD_ProcessDescriptor():

This function processes all the descriptor except CA_Descriptor. Take same arguments as above along with stream type to differteniate whether it is audio or video. Each descriptor is differentiated by its descriptor tag with as in ISO13818-1 and store them in their structures.

OPTIRET Decode_EMM():

This function decodes the EMM packets with the PID returned by Smart Card (in CA_Descriptor). EMM packets contains the management information for the subscriber.

OPTIRET Decode_CAT ():

This function copies complete CAT packet with all CA_Descriptor to EMM_INFO. Now this complete CAT packet will be parsed to Smart Card and Smart Card will select its CA_Descriptor and return to host.

OPTIRET Decode_ECM ():

This function decodes the ECM packet with the PID returned by SC (in CA_Descriptor). This packet contains the encrypted control words (CW's).Copy this packet in the structure which is supposed to parse to smart card.

OPTIRET Decode_PAT ():

This function decodes PAT. Each PAT contains the packet ID of each PMT packet along with their program numbers. If prog_num =0 , then this is for NIT and all other are for PMT packets. Store all this information in respective structures.

OPTIRET Decode_PMT ():

This function decodes PMT for each program. Take PMT_PID as argument. This PMT packet contains the PCR , Audio , Video PID along with their descriptors. This also contains CA_Descriptor which contains ECM PID. Now this audio , video, PCR PID is set for synchronization between audio and video.

Chapter 9

RESULTS AND DISCUSSIONS

Results of the project work carried out at HFCL, R&D Division where I was involved in the team of set-top Box project, as trainee are as follows:

One part of the project was to develop graphical user interface (GUI) also called Screen Display (OSD) for making the use of STB convenient and interactive various menus have been developed for this purpose using C/C++ which pop-up on the TV screen when the user wants to use STB along with the analog TV. The user can make his/her choices while navigating through the menus. Also he/she can make his/her personal settings such as setting the passwords. Some of these menus that have been developed together with the project team are shown in the chapter 5.

Second part was to extract the packets from the packetized transport stream which is MPEG encoded and to decode them and present them on the TV screen.Transport stream is fed to the STB through the cable which is containing TS packets containing program specific information in the form of PSI tables and then setting filters and extracting them.

Chapter 10

CONCLUSION

Major obstacle in Set-top Box is to develop a common conditional access (CA) system. Each broadcaster generally defines its own CA due to security reasons. However, there is increasing realization around the world that viewers should be able to receive all digital programs from one STB only. This manifestation is concrete decision policy in a number of countries. DVB consortium in Europe, USA and Japan has likewise declared the goal to provide an open solution, enabling multiple service providers to operate through a compatible cost effective receiver at home. For the same, it is important to keep the CA function as a module and provide a standard interface for building into a common system as needed.

CA system is user-friendly at any stage in the transaction e.g. it does not require any special action when changing channel and no significant delay in displaying TV signal. CA system must be effective in preventing piracy i.e. unauthorized viewing by users who are not entitled access to particular program or service. Smart cards must be resistant to tampering.

REFERENCES

[1] Jain P.C. , et al , "Digital satellite , cable and terrestrial Set-top Box with conditional access system", Proc. of National conference. on Communication (NCC-2003), IIT Madras , Feb. 2003.

[2] Jain P.C. , Mitra V. , "Digital Set-top Box – The State of the Art ", Proc. International conference and exhibition on terrestrial and satellite broadcasting, BES Review, vol. VIII , no.2 , April- June 2002 , pp 70-75.

[3] Jain P.C. et al " Low cost Interactive cable television Set-top Box" , Proc. of conference on affordable telecom and IT solutions for developing countries, IIT Chennai , Feb-March 2000.

[4] Driscoll G.O., " Essential Guide to Digital Set-top Box & Interactive TV ", pub. PHI PTR, ed. 1999.

[5] Mead D.C., " Direct Broadcast Satellite Communications", pub. Addison Wesley Wireless Communication Series, ed. 2000.

[6] Fujitsu OSD Functions Manual

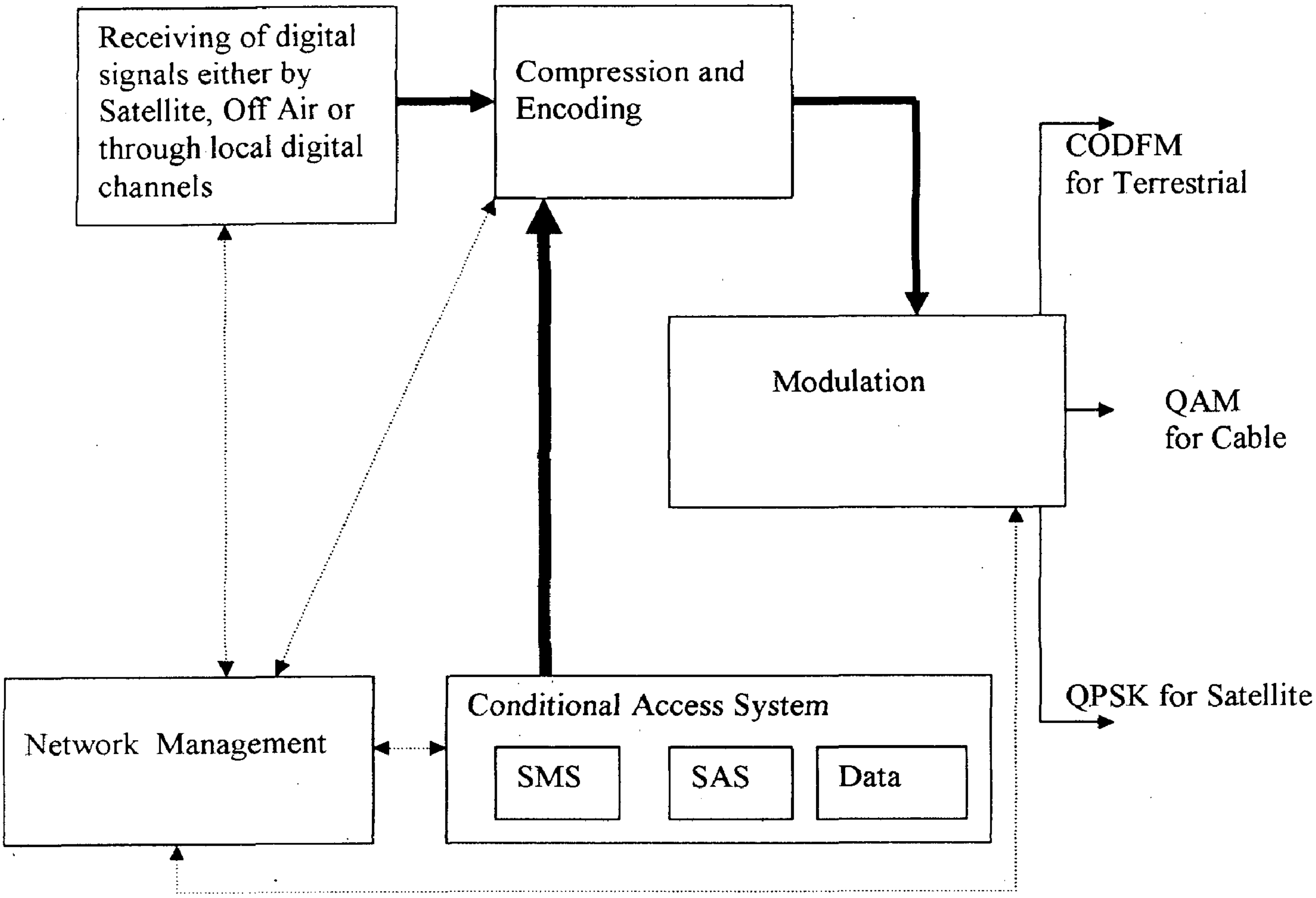
[7] MPEG-2 Transport Stream Manual , ITU/IEC 13818-1

[8] www.conax.com

[9] www.tektronix.com

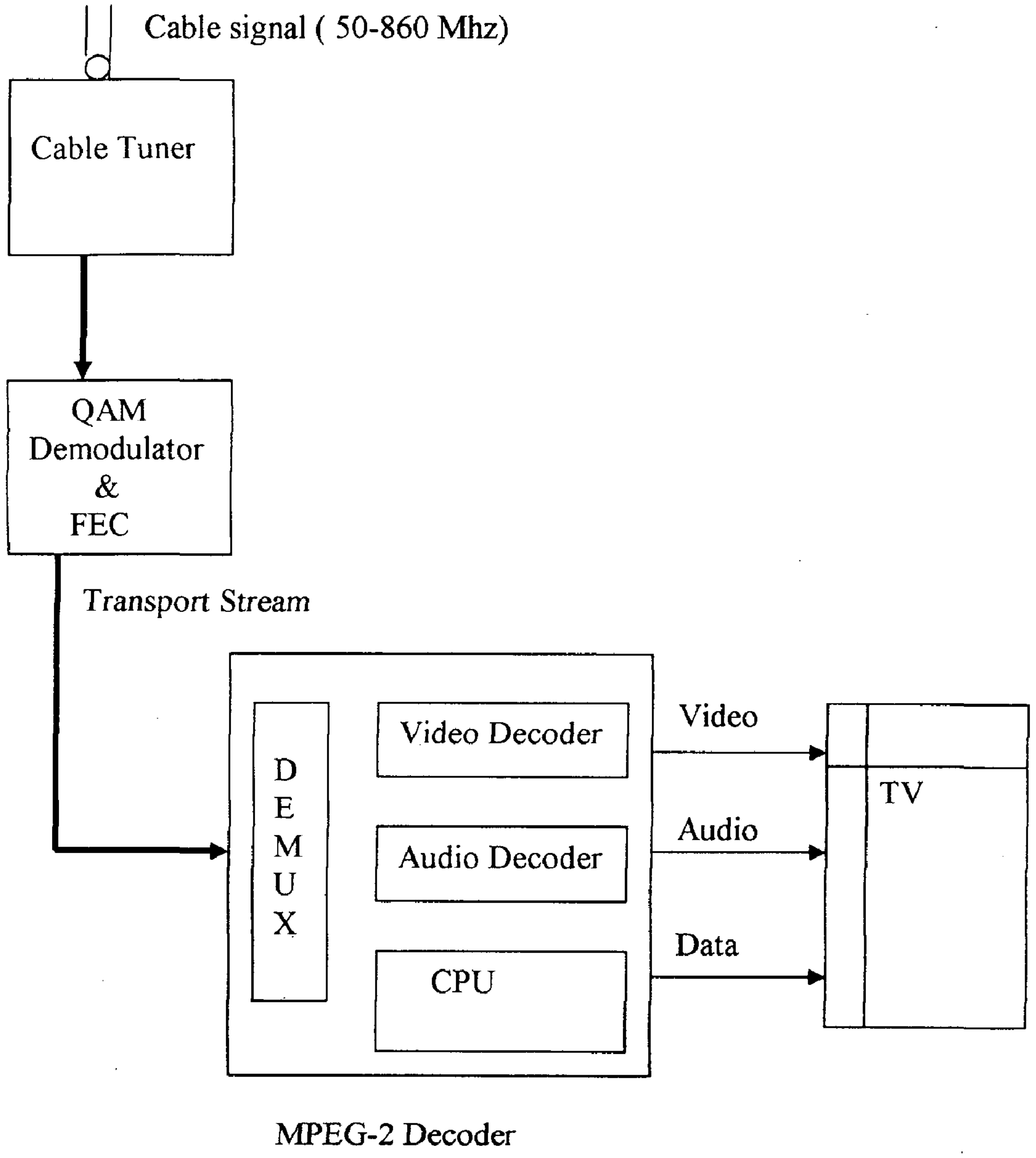
[10] http://canalplus-technologies.com

Appendix A


Receiving of digital signals either by Satellite, Off Air or through local digital channels
Compression and Encoding
CODFM for Terrestrial
Modulation
QAM for Cable
QPSK for Satellite
Network Management
Conditional Access System
SMS
SAS
Data

Simplified block diagram of a digital broadcasting system

Appendix B



Digital Set-top Box for Cable