CHAPTER

1

INTRODUCTION

The project **Universal Serial Bus** (**USB**) **Data Logger** is undertaken as a Final Year Project of Bachelor of Engineering (**B.E.**) course in Electronics Engineering department of Vivekanand Education Society's Institute of Technology (**V.E.S.I.T.**), Chembur, Mumbai. The project finds application in the High Energy Physics Department of Tata Institute of Fundamental Research (**T.I.F.R.**), Mumbai.

The project, as the name suggests is a general Data Acquisition System. The data or more precisely the analog signal is generated by 8 analog sources which are referred in the project as channels. The analog signal is converted to equivalent digital signal by using Analog to Digital Converter (**ADC**). The digital data is transmitted to the computer through USB Port for the purpose of monitoring.

The project involves development in three major areas which are as follows:

↳ THE PERIPHERAL HARDWARE:

The hardware at the Peripheral side must include intelligent systems in order to perform its intended duties dictated by the Specification. Hence an Embedded controller (Microcontroller) with USB capability or Serial Interface Engine (**SIE**) needs to be used that will take care of the requests from the host and interpret the received information in a usable form in order perform any predefined function such transmitting the information to some other device or control a real time application.

↳ THE PERIPHERAL SOFTWARE:

The Embedded controller is a piece of junk without its firmware which is truly the intelligence of the device. The firmware detects the communication targeted towards it from the host and responds with the necessary information. The functionality of the

peripheral hardware is strongly determined by the firmware which is required throughout the operation of the system; hence, the firmware is stored in the relatively permanent memory (EPROM, EEPROM, Flash memory). As it the code that hardly changes (it is Firm), it is aptly called Firmware.

↳ THE HOST COMPUTER SOFTWARE:

The software residing with the host computer (also referred as the device driver with an application program) must ensure that the Operating System knows how to communicate with the device which has been detected and to manage the data flow over the bus with additional duties assigned by the Specification (such as error checking). Under Windows, any communication with a USB peripheral must pass through a device driver that knows how to communicate both with the systems USB drivers and with the applications that access the device.

CHAPTER

2

PROBLEM DEFINITION

It was assumed that there are 8 channels generating analogue signal which needs to be converted from analogue form to digital form and to be transmitted to the host computer over USB port. The analogue signal is assumed to be available at the input connector of the board, which is in the range of 0 to 5 Volts with proper signal conditioning; so that there is no need of signal conditioning. Also the signal generation is not very rapid and input is slowly varying signal. Hence, the sampling of data need not be very fast and low sampling frequency can be employed.

An application when executed needs to display the status of each channel in a window on the host computer for the monitoring purposes. It was assumed that the value displayed for each channel should be a decimal value equivalent to the 8 bit binary data which is proportional to analogue signal value at the sampling instant. There should be 8 boxes with facility to clear the values in the boxes. The status of USB port to which the device is attached has to be shown in a box while providing facility to search the device according to the description, serial number as well as the device number and/or to display above mentioned parameters for the selected port. An extra facility needs to be given in order to log the data values of each channel in a text file from a given time instance for a specific duration. A Refresh button has to be provided in the application window so that fresh value or the current status of the channel is displayed. And lastly, an exit button should be provided in order to terminate the execution of application program to stop monitoring the channels.

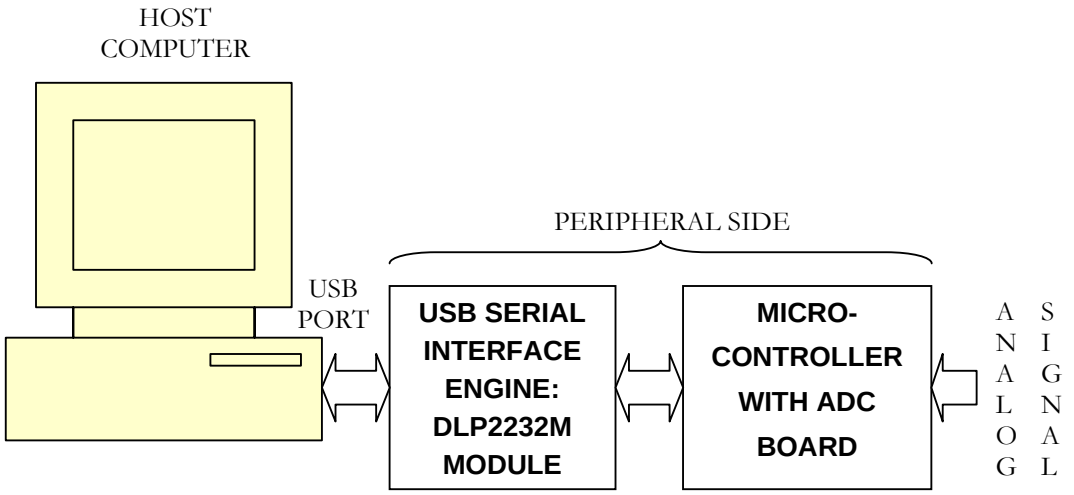
The system that is to be designed has to perform only the monitoring action with no need of feedback control. The system is naturally a measurement system and not a closed loop control system; hence, is a stable system.

CHAPTER

3

SYSTEM PERSPECTIVE

The project, when viewed from system perspective can be broken into three subsystems viz. host computer, Serial Interface Engine (SIE) and Microcontroller board as shown below:



USB DATA LOGGER – SYSTEM PERSPECTIVE

3.1 HOST COMPUTER

The Host Computer is a Personal Computer (PC) or other computer contains two components: a host controller and a root hub. These work together to enable the Operating system to communicate with the devices on the bus. The host controller formats data for transmitting on the bus and translates received data to a format the operating system components can understand. It also performs other functions related to managing communication on the bus. The root hub has one or more connectors for attaching devices. The root hub in combination with the host controller, detects the

attachment and removal of devices, carries out request from the host controller, and passes data between devices and the host controller.

The devices are the peripherals and additional hubs that connect to bus. A hub has one or more ports for connecting devices. Each device must contain circuits and code that knows how to communicate with the host. The specification defines the cables and connectors that connect devices to hubs.

3.2 SERIAL INTERFACE ENGINE (SIE)

The DLP2232M Module developed by Don L. Powrie (**DLP**) works as a SIE. The DLP-2232M utilizes Future Technology Devices International's (**FTDI**'s) third-generation USB Universal Asynchronous Receiver Transmitter (**UART**) / First In First Out (**FIFO**) I.C., the **FT2232C**. This low-cost development tool features two Multi-Purpose UART/FIFO controllers that can be configured individually in several different modes.

A single downstream USB port is converted to two IO channels that can each be individually configured as an UART interface, or a FIFO interface, without the need to add a USB hub. There are also several new modes which can be enabled in the external EEPROM, or by using Dynamic Link Library (**DLL**) driver commands. The FT2232C driver also incorporates the functions defined for FTDI's D2XX drivers, allowing applications programmers to interface software directly to the device using a Windows DLL.

3.3 MICROCONTROLLER BOARD

The microcontroller board receives signals from eight analog sources as input to 8-channel Analog to Digital Converter (**ADC**). The ADC is interfaced with the Microcontroller which initiates the conversion and monitors for the end of conversion. Then write the digital values corresponding to the entire 8 channel signals into the FT2232C FIFO RAM. The board also has a power supply circuit which powers the SIE board and itself.

Detailed description about these entities is followed in the subsequent chapters with all the relevant information such as Software code and Integrated Circuits (**IC**) datasheets given in the Appendix.

CHAPTER

4

ENTITY PERSPECTIVE - PERIPHERAL

From entity perspective, the peripheral side hardware can be separated into two subsystems viz. the USB Serial Interface Engine (SIE) and Microcontroller board. The communication between these two boards takes place on Request – Handshake basis.

4.1 SERIAL INTERFACE ENGINE (SIE)

The DLP-2232M module is a Single board, USB Dual Channel Serial / Parallel Ports with a variety of configurations. A single downstream USB port is converted to two IO channels that can each be individually configured as an UART interface, or a FIFO interface, without the need to add a USB hub.

The DLP-2232M features a quality four-layer printed circuit board with a solid ground plane, an integral 93C56 EEPROM on board for easy OEM customization and a standard 40-pin, 0.6in wide footprint. Integral power control and on-board MOSFET power switch make the DLP-2232M a perfect choice for USB bus-powered, high-power designs as well as self- and low-powered products.

There are also several new modes which can be enabled in the external EEPROM, or by using DLL driver commands. These include Synchronous Bit-Bang Mode, a CPU-Style FIFO Interface Mode, a Multi-Protocol Synchronous Serial Engine Interface Mode, MCU Host Bus Emulation Mode, and Fast Opto-Isolated Serial Interface Mode. Additionally, a new high output drive level option means that the device UART / FIFO IO pins will drive out at around three times the normal power level, allowing the data bus to be shared by several devices.

Classic BM-style Asynchronous Bit-Bang Mode is also supported, but has been enhanced to give the user access to the device's internal RD# and WR# strobes. FTDI provides a royalty free Virtual Com Port (VCP) driver that makes the peripheral ports look like a standard COM port to the PC. Most existing software applications should be able interface with the Virtual Com Port simply by reconfiguring them to use the new ports created by the driver. Using the VCP drivers, an application programmer would communicate with the device in exactly the same way as they would a regular PC COM port - using the Windows VCOMM API calls or a COM port library.

The FT2232C driver also incorporates the functions defined for FTDI's D2XX drivers, allowing applications programmers to interface software directly to the device using a Windows DLL.

The block diagram of the board and its description is as follows:

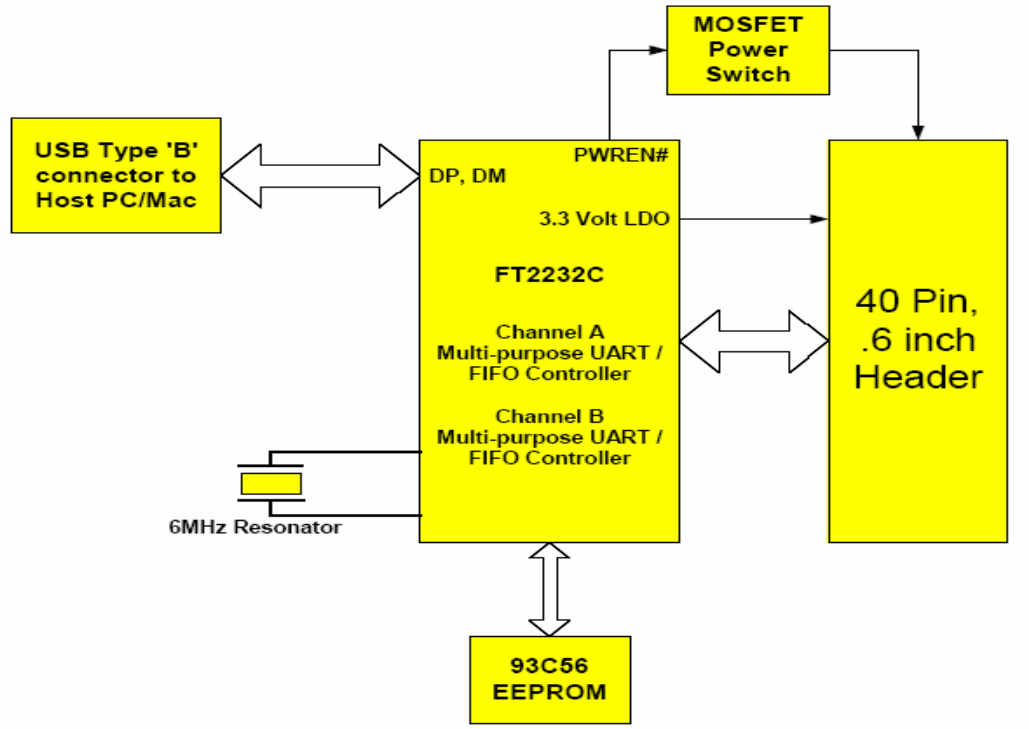
4.1.1 6MHz Oscillator

The 6MHz Oscillator cell generates a 6MHz reference clock input to the x8 Clock multiplier from an external 6MHz ceramic resonator.

4.1.2 Multi-Purpose UART / FIFO Controllers

The Multi-purpose UART / FIFO controllers handle the transfer of data between the Dual Port RX and TX buffers and the UART / FIFO transmit and receive registers. When configured as a UART it performs asynchronous 7/8 bit parallel to serial and serial to parallel conversion of the data on the RS232 (RS422 and RS485) interface. Control signals supported by UART mode include RTS, CTS, DSR, DTR, DCD and RI. There are also transmitter enable control signal pins (TXDEN) provided to assist with interfacing to RS485 transceivers. RTS/CTS, DSR/DTR and X-On/X-Off handshaking

options are also supported. Handshaking, where required, is handled in hardware to ensure fast response times. The UARTs also support the RS232 BREAK setting and detection conditions.



4.1.3 EEPROM Interface

The on-board 93C56 EEPROM allows each of the DLP-2232M module's channels to be independently configured as a serial UART (232 mode), or a parallel FIFO (245 mode). The EEPROM is used to enable the CPU-style FIFO interface, and Fast Opto-Isolated Serial interface modes. The EEPROM can also be used to customize the USB VID, PID, Serial Number, Product Description Strings and Power Descriptor value of the DLP-2232M for OEM applications. Other parameters controlled by the EEPROM include Remote Wake Up, Isochronous Transfer Mode, Soft Pull Down on Power-Off and USB 2.0 descriptor modes.

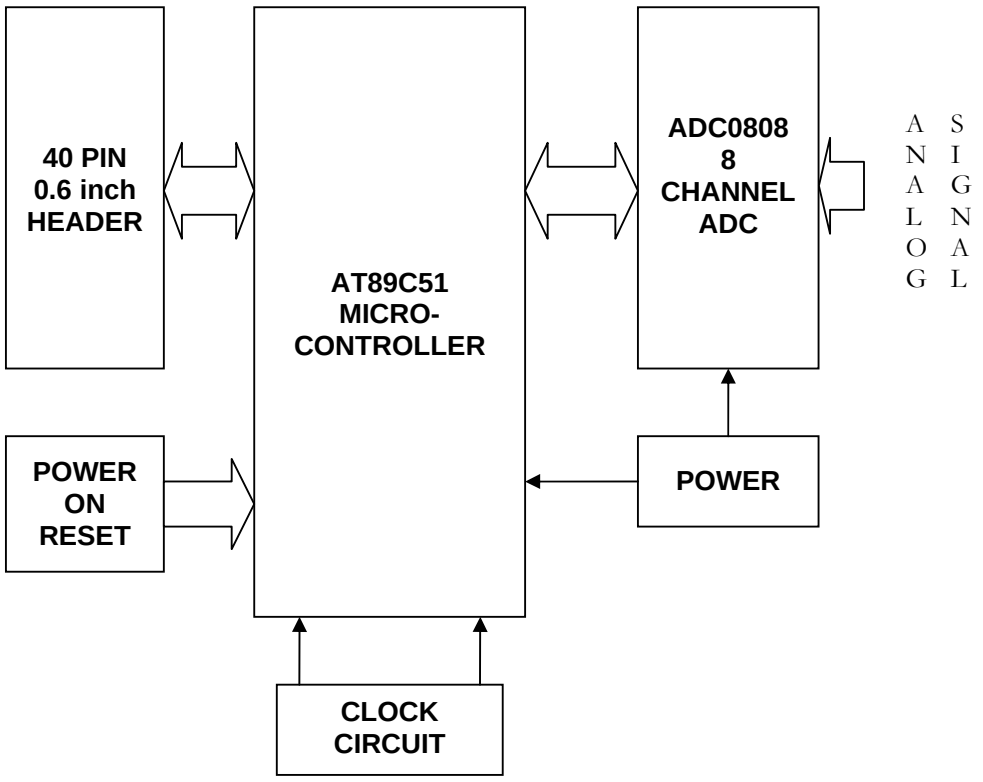
The schematic of the board is shown in APPENDIX A.1. The USB type B connector can be inserted into the connector provided on the board. The 40 pin (20x2) PCB connector is used to connect the board with the Microcontroller board.

4.2 MICROCONTROLLER BOARD

This board consists of AT90C51 Microcontroller interfaced with 8 Channel Analog to Digital Converter (**ADC**) ADC0808. The block diagram of the board and its functional description is as shown below:

4.2.1 Microcontroller:

AT89C51 40 pin Microcontroller is used to control ADC and read data to transfer it to the FT2232M chip's FIFO RAM. This Microcontroller has four 8 bit ports, hence total of 32 port lines. One of the ports (Port 0) is used as data bus between the DLP2232M Module and the Microcontroller. The handshake signals for the DLP2232M Module are generated from some of port 1 pins. The other port (Port 2) is configured as data bus between the ADC0808 and Microcontroller with some lines from port 1 and some from port 3 providing control signals for the ADC.



MICROCONTROLLER WITH ADC BOARD - BLOCK DIAGRAM

The Microcontroller initiates the Analog to Digital (**A to D**) conversion and selects appropriate channel using address lines. It also provides the clock for the A to D conversion and at the same time monitors for the End of Conversion. As soon as it detects End of Conversion, it reads the digital data from ADC and store the status of the channel data into its own RAM. The Microcontroller performs similar task for other channels by incrementing the address by 1 in sequential manner. When all the 8 channels are read and data is ready Microcontroller strobes data into the DLP2232M Module's FIFO RAM so that it can be read by the PC through the Device Driver.

4.2.2 Analog to Digital Converter:

The ADC is 8 bit Microprocessor Compatible A/D Converters with 8-Channel Multiplexer. The ADC0808 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A to D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals. The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE® outputs.

4.2.3 Power circuit:

Power circuit delivers a constant voltage of +5 V to the circuit. It can be constructed from any traditional rectifier circuit followed by a suitable filter with an appropriate voltage regulator. Or in some applications it can be directly connected to the DC regulated Power Supply, readily available in laboratories.

4.2.4 Reset Circuit:

The reset circuit is made up of two resistors, a capacitor & a switch. Upon power-up the capacitor charges from 0V to the supply voltage but due to finite time constant RC it takes some finite time for V_C (voltage across capacitor) to rise above

0.8V. This voltage is given to the Reset input of the microcontroller; hence the microcontroller gets reset when the power is just switched on & the voltage at the Reset pin is below V_{OL} level. But within a short span of time (5RC), V_C reaches +5V so that microcontroller can resume its normal operation. The circuit indeed works as a Power on Reset circuit. The switch is provided to discharge the capacitor, so that the microcontroller can be reset manually at any time.

4.2.5 Clock Circuit:

The clock circuit consists of a Crystal Oscillator operating at a desired frequency which generates clock signal with 50% duty cycle required by the microcontroller.

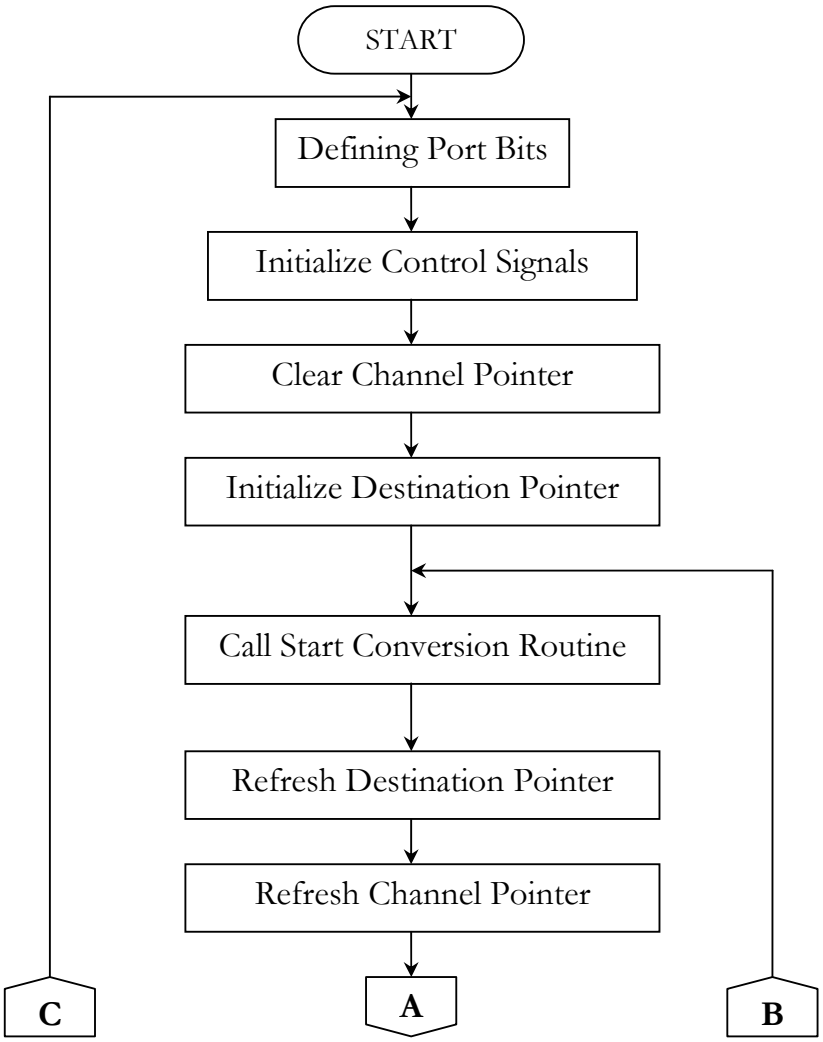
The Microcontroller board schematic is given in APPENDIX A.2. The firmware for the microcontroller is the intelligence of any embedded system. Hence, the firmware is explained in next chapter while the actual code is given in APPENDIX C.

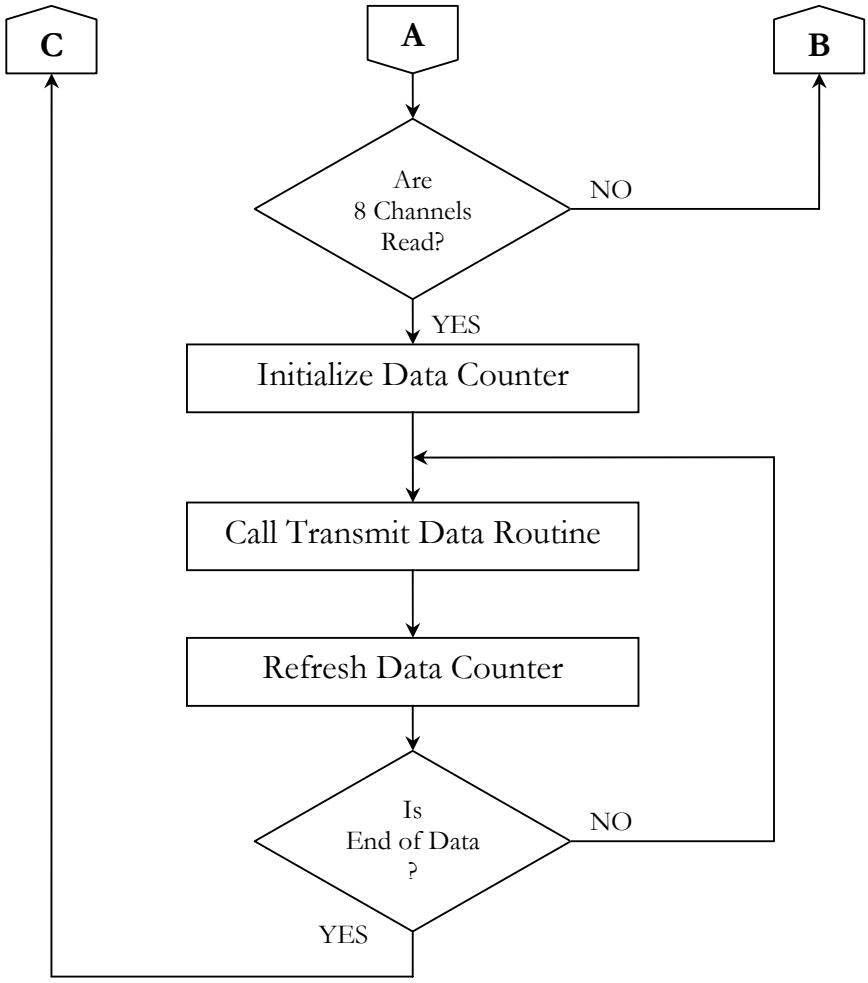
CHAPTER

5

ENTITY PERSPECTIVE – FIRMWARE

The firmware running on the Microcontroller enables the peripheral hardware to perform its intended task. The firmware is continuous loop, which keeps on executing indefinitely. The flowchart for the main program is given below:

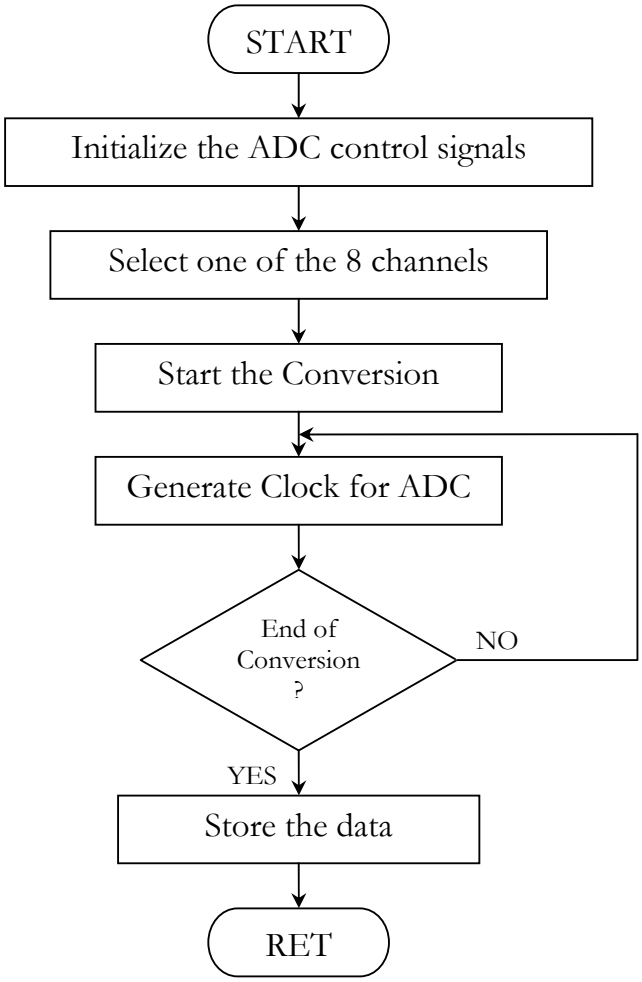




As seen, this the main program or Operating System code in Embedded Systems terminology, which executes all the time as long as systems is powered. The main program calls two subroutines viz. '**Start Conversion**' and '**Transmit Data**'.

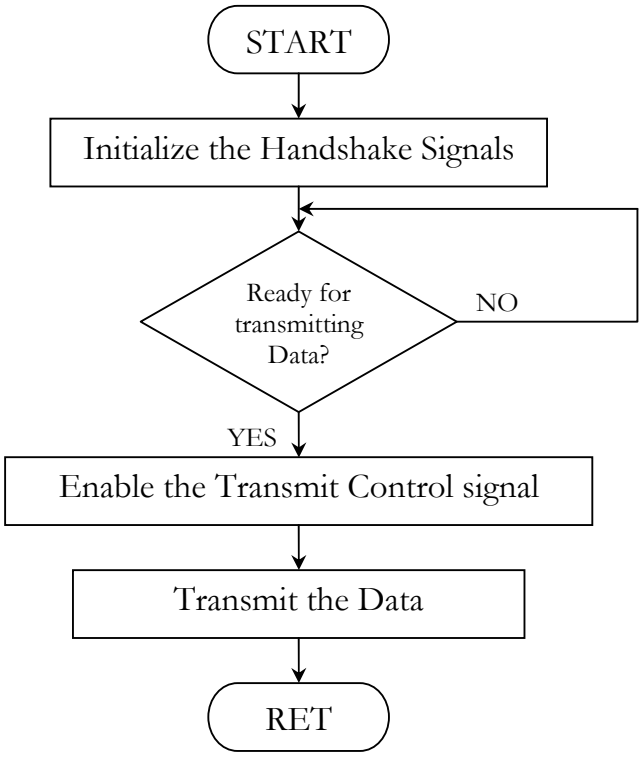
5.1 START CONVERSION ROUTINE

This subroutine initiates the Analog to Digital conversion process by making appropriate signal line active while selecting an analog channel for conversion. The clock waveform required for the ADC operation is generated while monitoring the End of Conversion. As soon as the End of Conversion is detected, so that the digital data is ready to be read by the Microcontroller, it stores the data value in scratchpad RAM area of the Microcontroller.



5.2 TRANSMIT DATA ROUTINE

This subroutine transmits the digital data stored in scratch pad RAM area pointed by the Destination pointer to the DLP2232M Module. The FT2232 SIE then inserts the data received from the Microcontroller into its FIFO RAM.



The actual assembly language code for the firmware is given in APPENDIX C.

CHAPTER

6

ENTITY PERSPECTIVE – HOST COMPUTER

From entity perspective, the host computer is a machine in which the software code runs. For the USB system to work it is necessary that the host machine have a host controller and root hub. The host computer allows software code to be executed in order to perform the intended function. The software is basically used for monitoring the various parameters like temperature, pressure, gas flow etc. The host computer reads the contents from the FIFO RAM of the DLP USB 232 Kit which in turn reads the contents from the Microcontroller 89C51. A programming language and development environment on the host are thus required for writing and debugging the host software.

The host software may include a device driver and application code.

6.1 DEVICE DRIVER:

To write a device driver Visual C++, Visual Basic or Delphi are required, which are capable of compiling the Win32 Driver Model (**WDM**) drivers needed for the USB devices. Since we are using the FTDI USB chip FT2232C, writing the drivers is not a major concern as royalty free drivers are provided by the chip itself. The drivers can run in various operating systems such as Windows 98, Windows ME, Windows 2000, Windows XP, Linux, Mac OS-X and Windows CE.Net. In the case of the FT2232C, the driver type is selected by a setting in the EEPROM.

The drivers available on the FT2232C Device are VCP and D2XX drivers.

6.1.1 VCP Drivers:

Virtual COM port (VCP) drivers cause the USB device to appear as an additional COM port available to the PC. Application software can access the USB device in the same way as it would access a standard COM port. In short, the VCP driver emulates a standard PC serial port such that the USB device may be communicated with as a standard RS232 device

6.1.2 D2XX Drivers:

D2XX drivers allow direct access to the USB device through DLL access. Application software can access the USB device through a series of DLL function calls. Thus, the D2XX driver allows direct access to a USB device via a DLL interface.

In our project, we are using D2XX drivers for writing the application programs as FTDI's "D2XX Direct Drivers" for Windows offer an alternative solution to VCP drivers which allow application software to interface with FT2232C Dual USB UART/FIFO using a DLL instead of a Virtual COM Port.

6.2 APPLICATION PROGRAMS

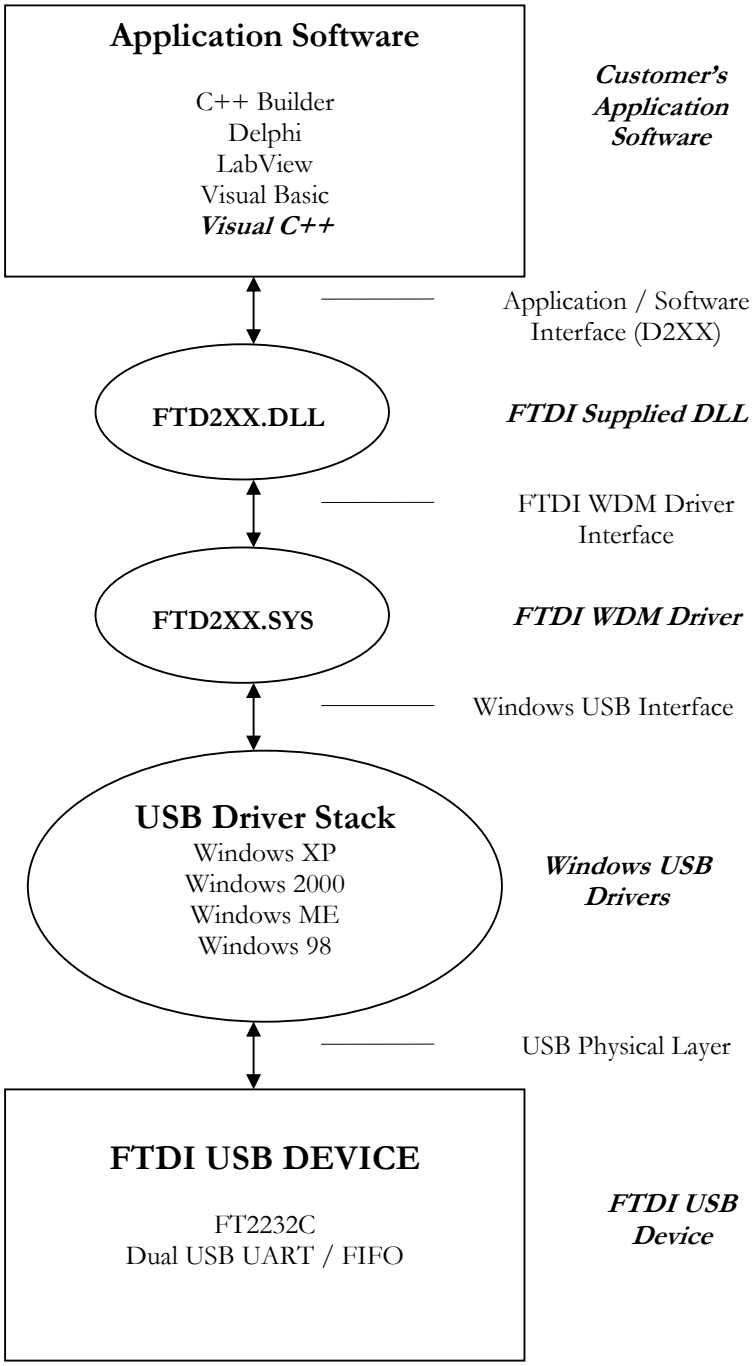
Application Programs are written in various languages like Visual C++, Visual Basic, C++ Builder, Delphi etc. As a language platform, Visual C++ was selected because of the following reasons:

- System I/O calls are provided
- Faster runtime which is essential when interacting with the hardware.

Apart from these, Visual C++ also provides good tools to create a DLL

The following flowchart shows how exactly customer's application software interacts with the FTDI USB device.

The architecture of the D2XX drivers consists of a Windows WDM driver that communicates with the device via the Windows USB stack and a DLL which interfaces the application software to the WDM driver.



Hierarchical view of the System as a Software Entity

The FTD2XX Programming Guide includes the following functions through which the contents of the device are accessed.

01. Classic Interface Functions : This explains general functions like Open, Read, Write, Close, ResetDevice, SetTimeouts, ListDevices, SetBaudRate etc

02. EEPROM Interface Functions: This allow application software to read/program the various fields in the EEPROM including the user defined area which can be used for application purposes.

03. Extended API Functions: This allows control of the additional features

04. FT-Win32 API Functions: This function is a sophisticated alternative to the Classic interface. Using this function existing Windows legacy Communication applications can be easily converted to use the D2XX interface by replacing the standard Win32 API calls with the equivalent FT – Win32 API calls.

6.3 FTDI FUNCTIONS USED IN THE PROGRAM:

6.3.1 FT_ListDevices

Get information concerning the devices currently connected. This function can return information such as the number of devices connected, the device serial number and device description strings, and the location IDs of connected devices.

6.3.2 FT_Open

Open the device and return a handle which will be used for subsequent accesses.

6.3.3 FT_OpenEx

Open the specified device and return a handle that will be used for subsequent accesses. The device can be specified by its serial number, device description or location.

6.3.4 FT_Close

Close an open device

6.3.5 FT_Read

Read data from the device. This function does not return until no of bytes to be read are read into the buffer.

6.3.6 FT_Write

Write data to the device. Read data from the device. This function does not return until no of bytes to be written are written into the buffer.

6.3.7 FT_ResetDevice

Send a reset command to the device

6.3.8 FT_SetTimeouts

This function sets the read and write timeouts for the device.

6.3.9 FT_GetStatus

Gets the device status including number of characters in the receive queue, number of characters in the transmit queue, and the current event status.

6.4 VISUAL C++

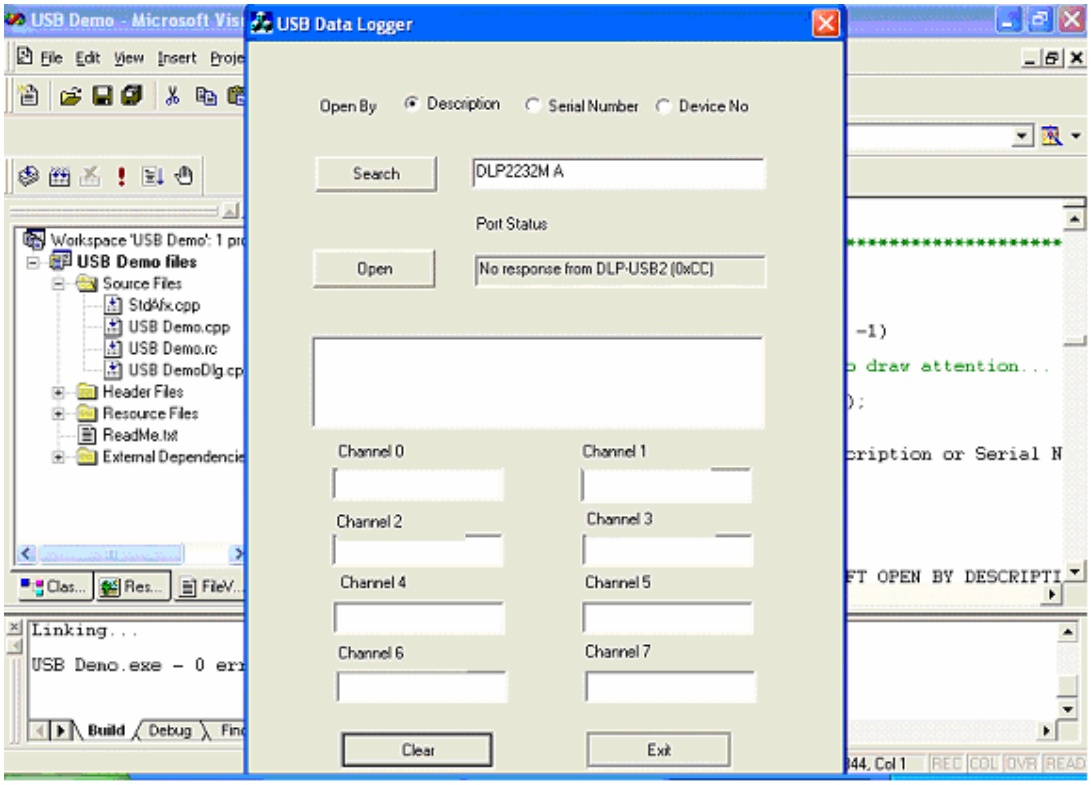
The application coding has been done in Visual C++ (VC++). Visual C++ incorporates a range of fully integrated tools designed to make the whole process of writing Windows programs easy. The Integrated Development Environment that comes with Visual C++ is a completely self contained environment for creating, compiling, linking and testing Windows programs. The fundamental parts of Visual C++ include the editor, the compiler, linker and the libraries. These are the basic tools that are essential for writing and executing a C++ program.

6.5 USB DATA LOGGER

The software as shown continuously monitors inputs from 8 different channels and displays it in various boxes.

When the application is run from the VC++ workspace, the project code gets compiled, linked and executable file is generated through which the application window as shown below appears on the screen.

The window shown below is snapshot of the application window that appears just after the application is run.



The software has options for

01. Searching the device
    a) By Description
    b) By Serial Number
    c) By Device No.
02. Opening the device
03. Monitoring 8 different channels
04. Clearing the contents of received data

The following options can be explained in details as follows:

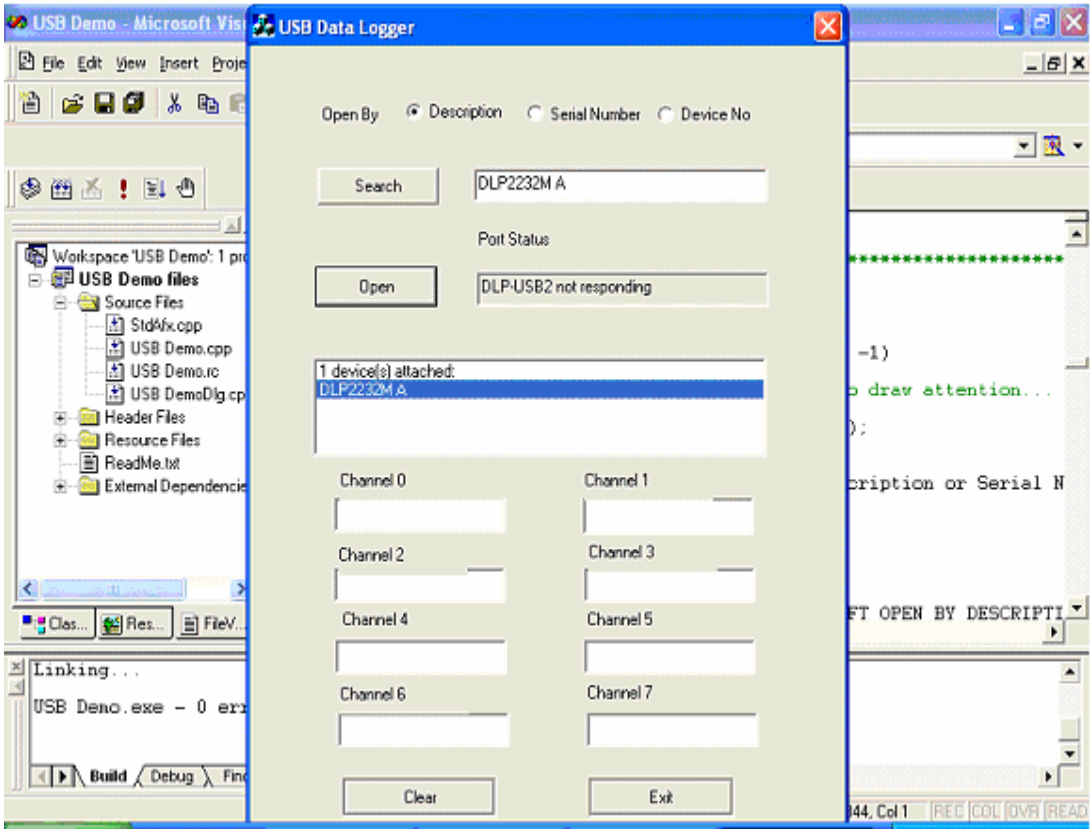
6.5.1 Searching the device

On Click of the Search Button and by selecting the appropriate radio button, the device can be selected by either description, serial number or device number.

The Search button detects all FTDI devices connected to the USB bus and lists each device by its description string, serial number or device number in the List box. When the string is selected in the List box, it appears on the edit box of the search button. The default value in the search edit box is “Enter the device description or serial number here”.

6.5.1.1 By Description:

The Description radio button selection will cause the Search function to locate all FTDI devices and present their description strings in the list box. The application window snapshot for this option is shown below.

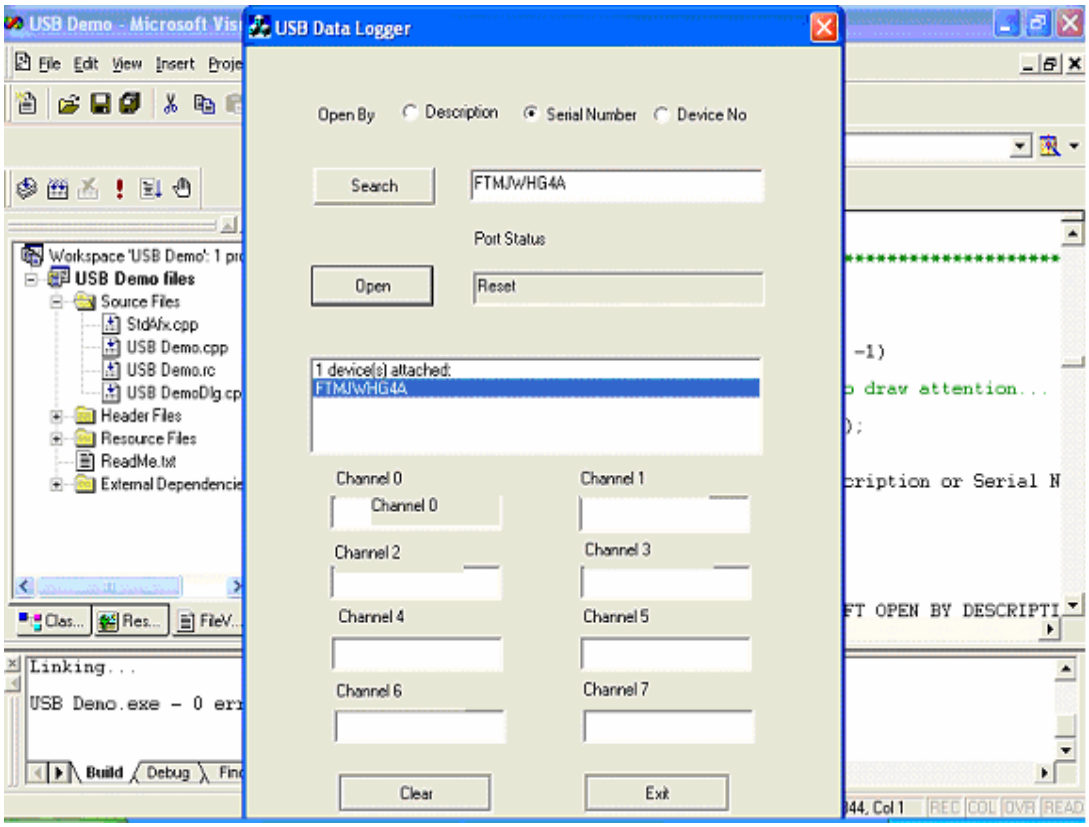


This mode assumes that an EEPROM device is connected to the FTDI Chip and that a description string has been written to the EEPROM device.

6.5.1.2 By Serial Number:

The Serial Number radio button selection will cause the Search function to locate all FTDI devices and present their serial numbers in the list box. This mode assumes that an EEPROM device is connected to the FTDI Chip and that a serial number has been written to the EEPROM device.

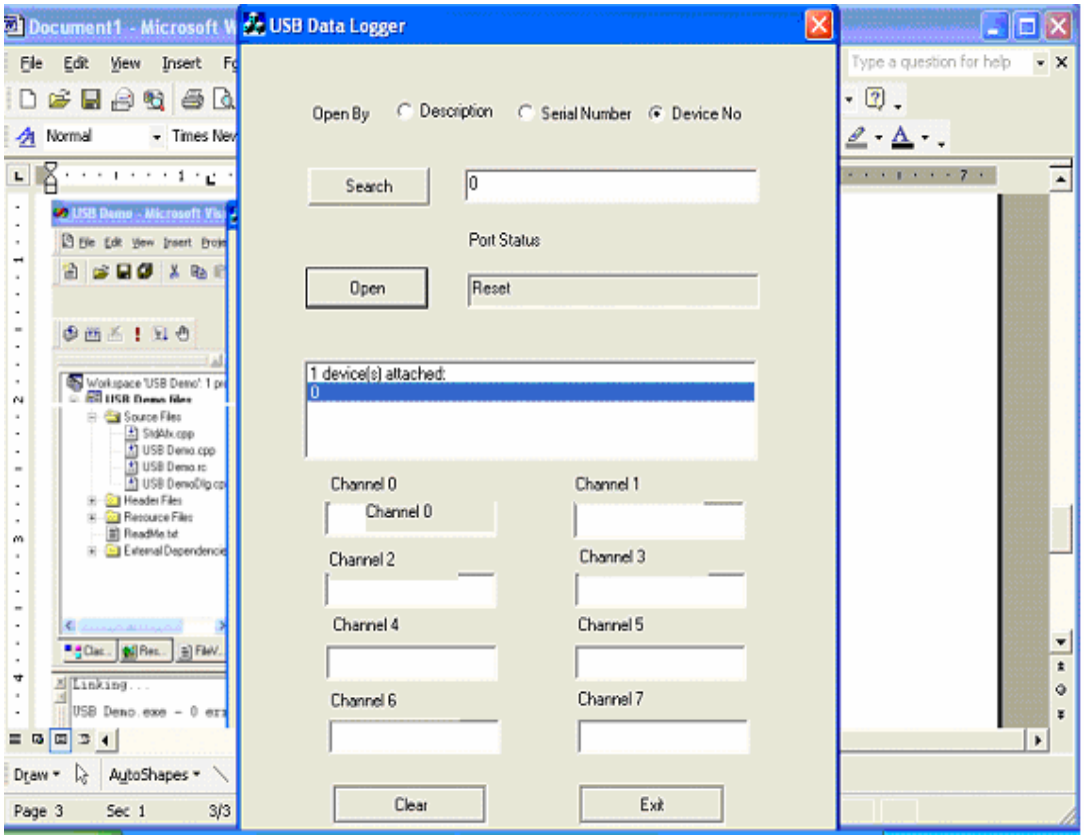
The application window snapshot for this option is shown below.



6.5.1.3 By Device Number:

The Device Number radio button selection will cause the Search function to locate all FTDI devices and present their device numbers in the list box. This mode works without having an EEPROM device connected to the chip. It simply places a device number in the list box for each FTDI Chip connected – in no particular order.

The application window snapshot for this option is shown below.



For the code of these functions Refer to APPENDIX D.1.

6.5.2 Opening the device

On Click of the Open Button, the device in the search edit box is opened by either description, serial number or device number

The Search edit box contains the description string, serial number or device number of the device that will be opened when the Open button is clicked. The desired device must be connected to the host PC or the Open function will report an error. The open edit box contains the current status (Open or Closed) of the FTDI device attached to the USB port.

For the code of these functions Refer to APPENDIX D.2.

6.5.3 Monitoring 8 different channels

On Click of the Open Button, the edit boxes corresponding to eight different channels display the respective values.

The channels are analog inputs from various transducers and signal conditioners. The parameters are temperature, pressure, flow etc which are been continuously monitored. The 8 edit boxes display the respective values i.e. data received from 8 different channels.

For the code of these functions Refer to APPENDIX D.3

6.5.4 Clearing the contents of received data

On Click of the clear button, all data in the List box and in 8 different channels are cleared.

For the code of these functions Refer to APPENDIX D.3

CHAPTER

7

APPLICATIONS

The application of the USB Data Logger project is basically in the detection of Resistive Plate Chamber* (RPC) *dark current monitoring.

The USB Data Logger project was designed so as to be used in the Department of High Energy Physics at TIFR. The department conducts various experiments in the field of neutrino physics.

7.1 NEUTRINO PHYSICS

Neutrinos are one of the fundamental particles, which make up the universe. Neutrinos are electrically neutral and were initially thought to be mass less. There are three types or flavors of neutrinos known. Recent evidences seem to indicate that neutrinos not only have mass, but also experience mixing among these flavors. This leads to the phenomena of neutrino oscillations that can explain the discrepancy between theory and observations about its flux.

Historically, the Indian initiative in neutrino physics experiments goes back to several decades. To exploit this valuable expertise for the emerging new area of observational neutrino physics, an idea to construct a neutrino detector at an **India-based Neutrino Observatory (INO)** was mooted a few years ago.

The proposed neutrino detector will use **Glass Resistive Plate Chambers (glass RPCs)** as active detector elements. It will comprise of 140 layers of horizontally arranged iron plates of 6cm thickness, interleaved with 2.5cm gap between successive layers of iron plates to house the glass RPCs.

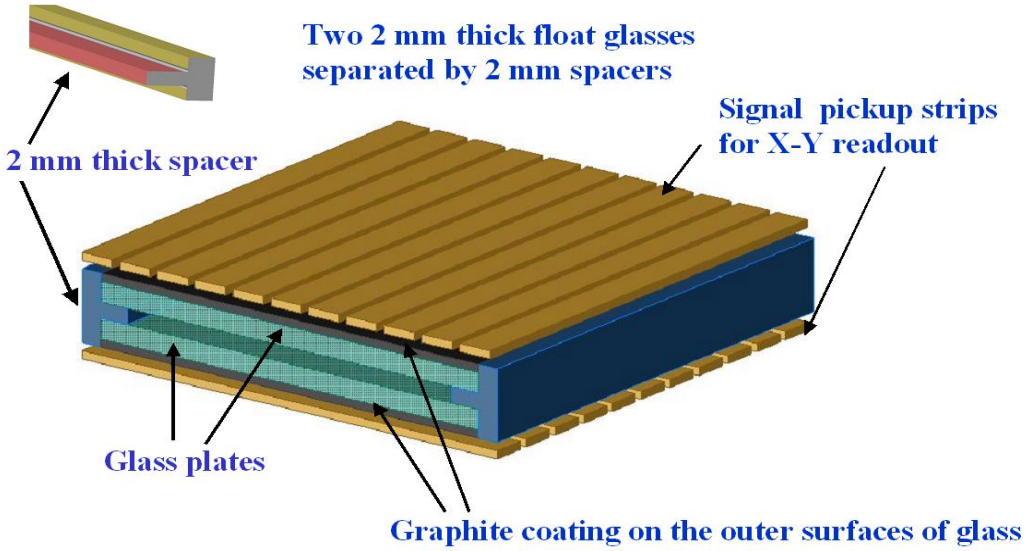
7.2 GLASS RPC - ACTIVE NEUTRINO DETECTOR

Glass RPCs are rugged and low-cost gas detectors which are being used or planned for extensively in a number of high energy and astro-particle physics

experiments. They find applications for charged particle detection, time of flight, tracking and digital calorimetry due to their large signal amplitudes as well as excellent position and time resolutions. A dedicated R&D programme is currently under way to develop and characterise these chambers, ultimately leading to their large scale production required for the INO detector.

7.2.1 GLASS RPC – Construction and Working

Glass RPC is a gaseous detector composed of two parallel electrodes made up of float glass The two electrodes, 2mm thick, are mounted 2mm apart by means of highly insulated spacers. A suitable gas mixture is flown at the atmospheric pressure through the gap while an appropriate electric field is applied across the glass electrodes through a resistive coating on their outer surfaces. An ionising charged particle traversing the gap, initiates a streamer in the gas volume that results in a local discharge of the electrodes.

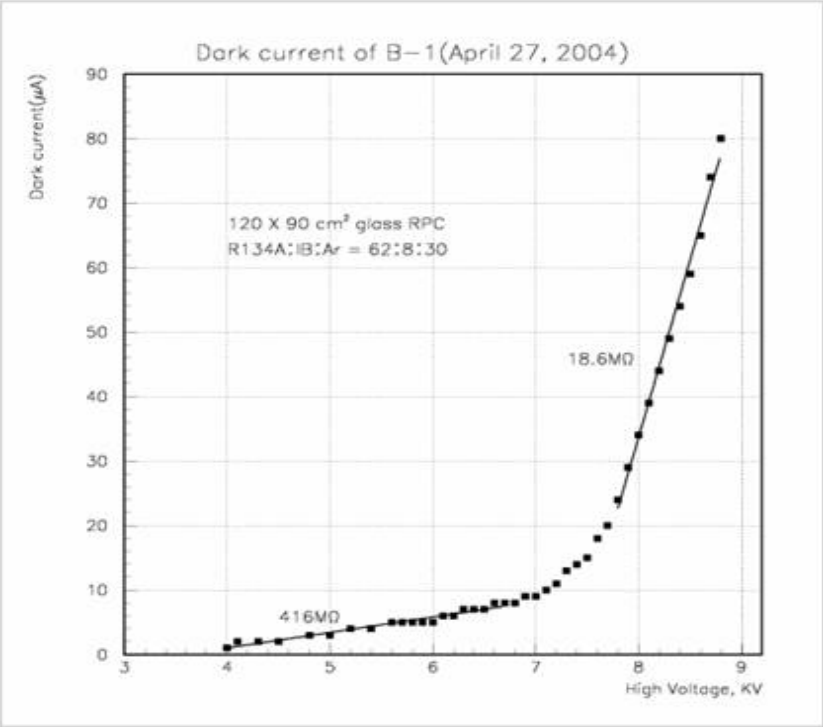


This discharge is limited to a tiny area due to the high resistivity of the glass electrodes and the quenching characteristics of the gas. The discharge induces an electrical signal on external pickup strips, which can be used to record the location and time of ionisation. The discharge is quenched when all of the locally available charge is

consumed. The discharged area recharges slowly through the high-resistivity glass plates. Typical signal amplitude is about 100-200mV across a 50Ohm load and its rise time is less than a nanosecond.

7.2.2 GLASS RPC: V- I Characteristics

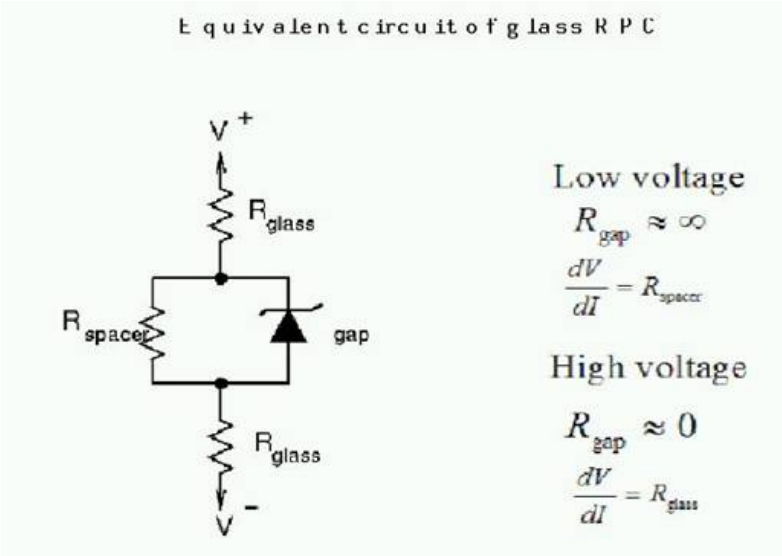
Glass RPCs have a distinctive and readily understandable voltage versus current relationship. At lower applied voltages, the chamber's gas gap offers very high resistance, hence its V-I characteristic in this region is mainly characterised by the insulative spacers placed between the electrodes. When the chamber starts operating in the active region, resistivity of the glass electrodes alone determines its characteristics. The V – I Characteristic is shown below.



7.2.3 Electrical Equivalent of RPC

It is essential to test all the glass RPCs which are mass produced to check whether its I-V characteristics confirm to the above behaviour. It is also important to monitor the high voltage and current through the RPC to make sure that the chamber is operating in stable manner. In fact, monitoring the chamber current is the most effective

way to monitor stability of the chamber during its operation. The high voltage and current of the chamber can be monitored using the monitor D.C voltage outputs provided by the high voltage power suppy which is used to bias the chamber.



Hence, the USB Data Logger tends to be useful in the monitoring of dark current through the RPC which in turn is a measure of the stability of the chamber.

Equally important is to monitor other ambient conditions of the lab in which RPC detectors are being operated in. These include parameters such as temperature, pressure, gas flow through the RPC etc.

Again, appropriate monitor D.C voltages from the transducers will be used to monitor these parameters. Combined, the data logger will provide invaluable information on the operating conditions of the RPCs and helps us analyse the data produced by the RPC in a much better manner.

CHAPTER

8

FUTURE DEVELOPMENTS

Apart from RPC dark current, temperature, pressure, gas flow and other parameter monitoring, the designed project finds itself useful in a wide number of applications. With few changes incorporated in the hardware and software, the project can be implemented in various areas of interest.

INSTRUMENTATION SYSTEMS:

USB data logger as a data acquisition system can be used in various industries for receiving the data from a real world process. In situations where it is necessary not only to acquire and process data but also to send back to the real time process, we have a data acquisition and control system. The following project is used only for monitoring the data but can be also used as a closed loop system in control system applications with some changes in the design.

SERIAL COMMUNICATION:

The designed project makes entire utilization of the USB to Parallel port conversion. As the project can be used to control any application on the peripheral side, serial communication can also be implemented so as to interface any device lacking USB port. Older (legacy) computers or laptops do not have provision of a USB port, so designing a USB to RS232C converter can thus be very useful.

The serial port implementation is possible in 2 different ways:

- **Using DLP USB Kit (Channel B)**

DLP USB Kit has 2 Channels A and B which can be independently configured as a serial or parallel port. By some sort of switching arrangement, it is possible to directly interface a Sipex SP21EHCA or Maxim MAX232 IC to the channel B. The

Sipex SP21EHCA IC which is a TTL to RS232C converter can thus provide a serial port at the output.

The schematic for the Serial Communication Interface for both ports of FT2232C using Sipex SP21EHCA is shown in APPENDIX A.3

- **Using Microcontroller AT89C51**

Microcontroller 89C51 has 2 port pins Rxd and Txd for serial communication. By using a single channel and eliminating the ADC at the output, serial signals can be interfaced to MAX232 IC (TTL-RS232C Converter) thus providing the required RS232 signals at the output.

Finally, the designed project can be extended further to monitor *n* different channels rather than 8 channels. This can be done by the addition of a multiplexer viz. 16:1, 32:1, 64:1 etc. Thus the range of the system can be extended so as to incorporate wide range of input data.

APPENDIX

A

SCHEMATICS

The Appendix contains the circuit diagrams of the boards mentioned in chapter 4 and 5 on the Entity Perspective – Peripheral. These are the schematics for the Peripheral side hardware.

A.1 DLP2232M SERIAL INTERFACE ENGINE (SIE)

The schematic for this board is given here. The block diagram explanation of the board is given in section 4.1 SERIAL INTERFACE ENGINE of chapter 4 ENTITY PERSPECTIVE – PERIPHERAL.

A.2 MICROCONTROLLER BOARD

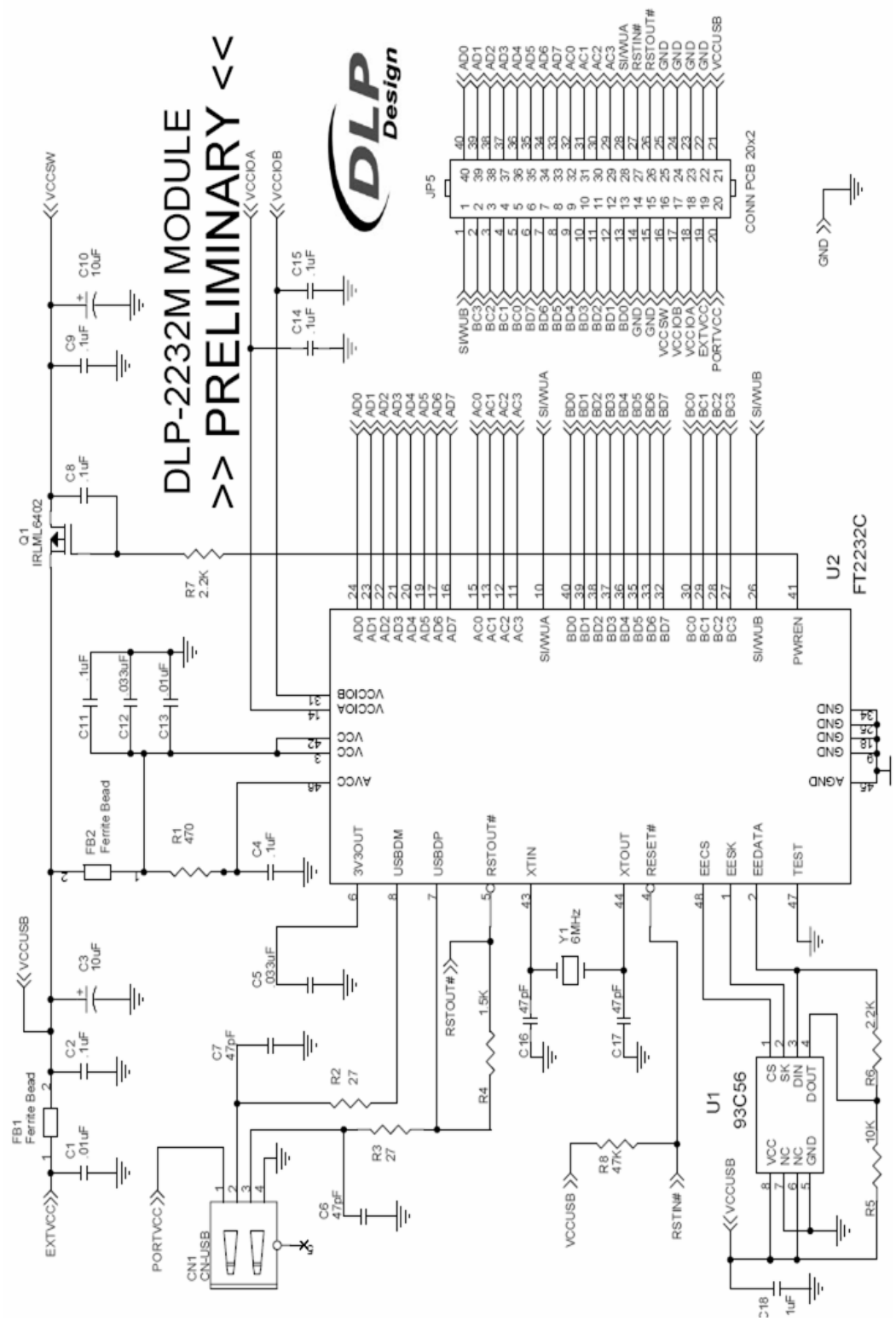
The schematic for this board is given here. The block diagram explanation of the board is given in section 4.2 MICROCONTROLLER BOARD of chapter 4 ENTITY PERSPECTIVE – PERIPHERAL.

A.2 SERIAL COMMINICATION (RS-232) BOARD

The schematic is proposed implementation of serial communication RS232 interface using Sipex SP21EHCA in the system so that the system can convert between two standards USB and RS232.

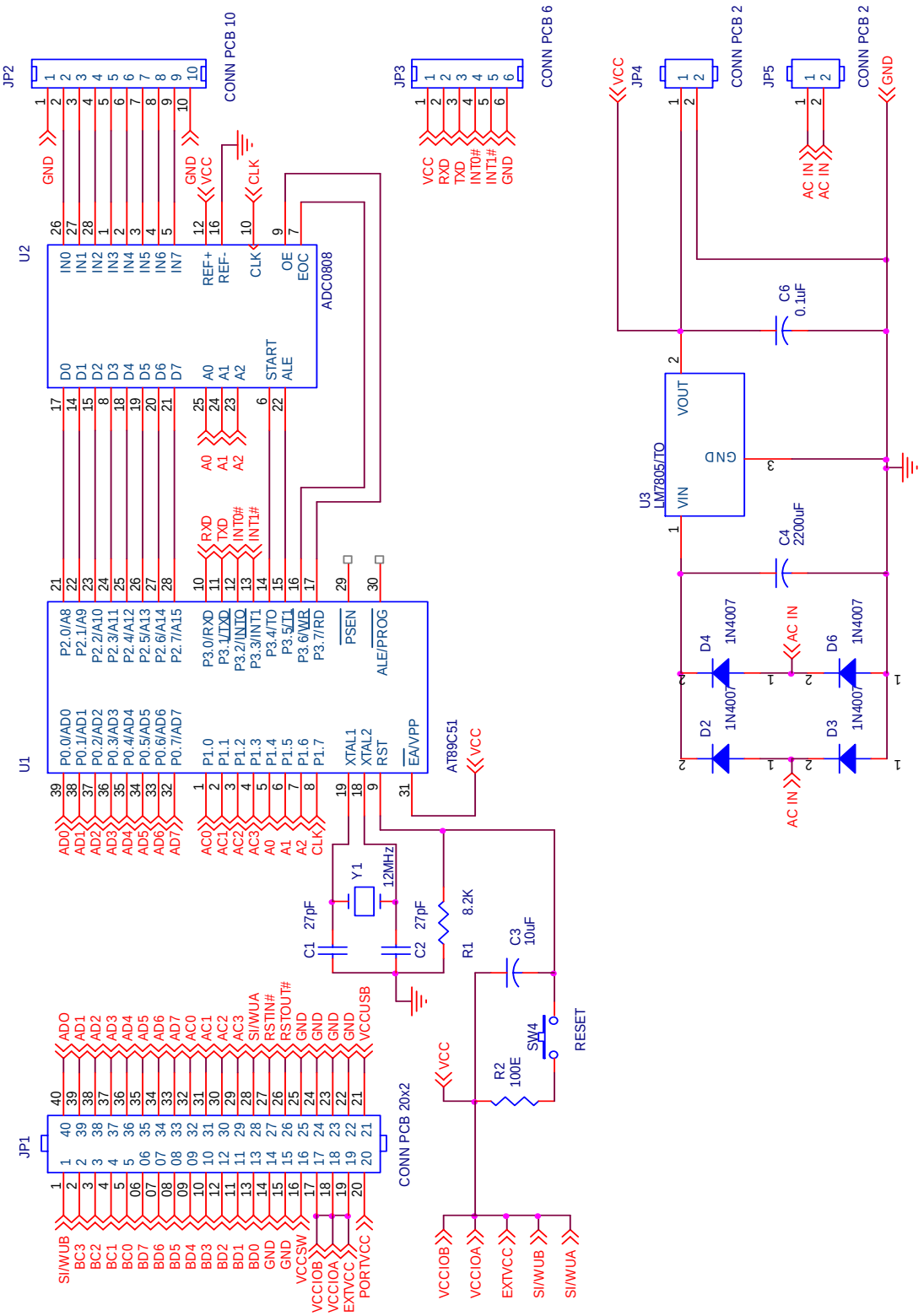
APPENDIX A.1

SERIAL INTERFACE ENGINE (SIE) BOARD



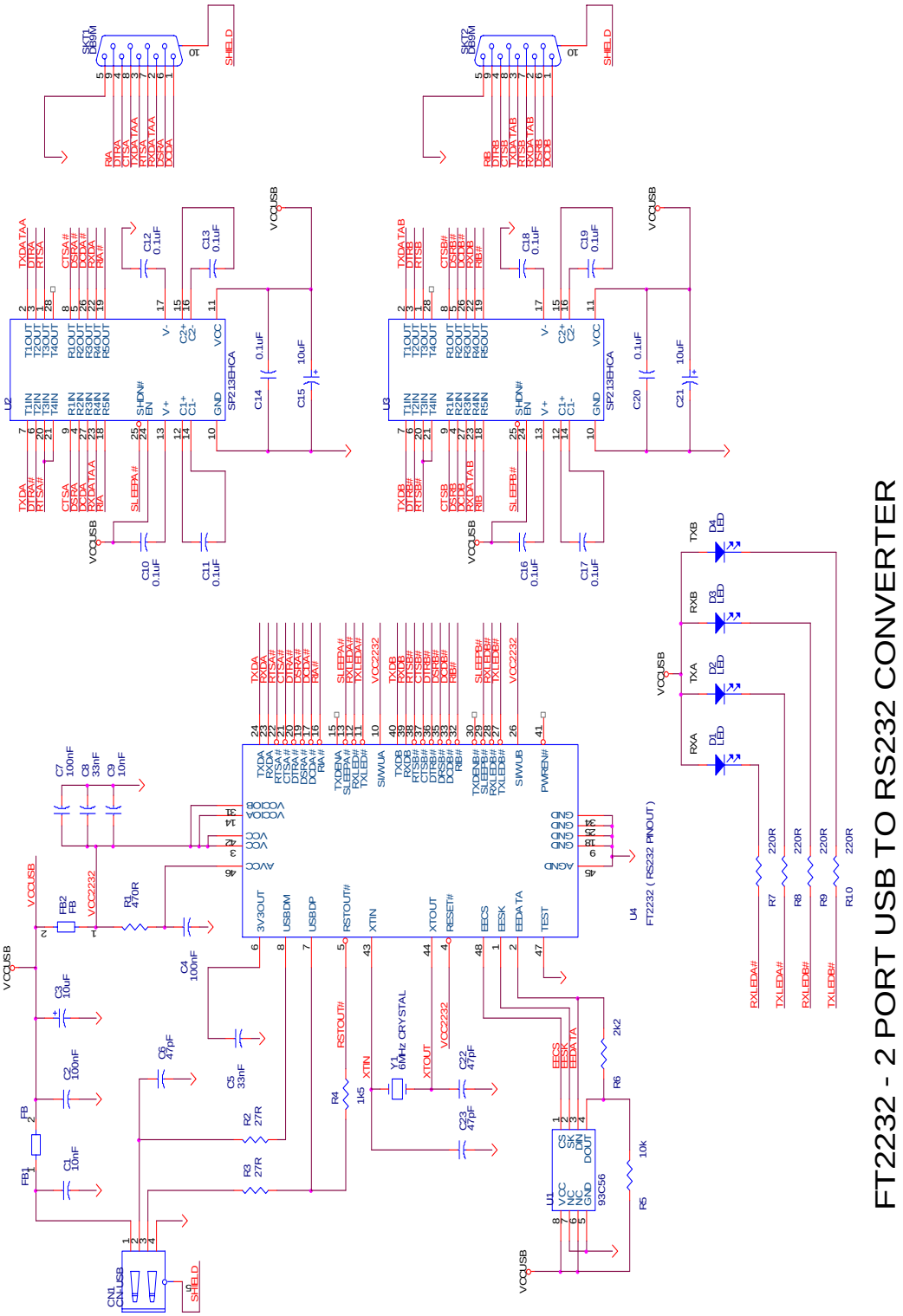
APPENDIX A.2

MICROCNTROLLER BOARD



APPENDIX A.3

SERIAL COMMINICATION (RS-232) BOARD



APPENDIX

B

MANUFACTURERS' DATA SHEETS

This appendix contains data sheets representative of Integrated Circuits (ICs) used in the Peripheral Hardware circuits.

B.1 FT2232C Dual USB UART / FIFO I.C.

B.2 AT89C51 Microcontroller

B.3 ADC0808 8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer

APPENDIX B.1

FT2232C Dual USB UART / FIFO I.C.

1.0 Introduction

The FT2232C is the 3rd generation of FTDI's popular USB UART / FIFO I.C. family. This device features two Multi-Purpose UART / FIFO controllers which can be configured individually in several different modes. As well as a UART interface, FIFO interface and Bit-Bang IO modes of the 2nd generation FT232BM and FT245BM devices, the FT2232C offers a variety of additional new modes of operation, including a Multi-Protocol Synchronous Serial Engine interface which is designed specifically for synchronous serial protocols such as JTAG and SPI bus.

1.1 Features Summary

HARDWARE FEATURES

- Single Chip USB ⇔ Dual Channel Serial / Parallel Ports with a variety of configurations
- Entire USB protocol handled on the chip...no USB-specific firmware programming required
- FT232BM-style UART interface option with full Handshaking & Modem interface signals
- UART Interface supports 7 / 8 bit data, 1 / 2 stop bits, and Odd / Even / Mark / Space / No Parity
- Transfer Data Rate 300 to 1 Mega Baud (RS232)
- Transfer Data Rate 300 to 3 Mega Baud (TTL and RS422 / RS485)
- Auto Transmit Enable control for RS485 serial applications using TXDEN pin
- FT245BM-style FIFO interface option with bi-directional data bus and simple 4 wire handshake interface
- Transfer Data Rate up to 1 MegaByte / Second
- Enhanced Bit-Bang Mode interface option
- New Synchronous Bit-Bang Mode interface option
- New CPU-Style FIFO Interface Mode option
- New Multi-Protocol Synchronous Serial Engine (MPSSE) interface option
- New MCU Host Bus Emulation Mode option
- New Fast Opto-Isolated Serial Interface Mode option
- Interface mode and USB Description strings configurable in external EEPROM
- EEPROM Configurable on board via USB
- Support for USB Suspend and Resume conditions via PWREN#, and SI / WU pins
- Support for bus powered, self powered, and high-power bus powered USB configurations
- Integrated Power-On-Reset circuit, with optional Reset input and Reset Output pins
- 5V and 3.3V logic IO Interfacing with independent level conversion on each channel
- Integrated 3.3V LDO Regulator for USB IO
- Integrated 6MHz – 48Mhz clock multiplier PLL
- USB Bulk or Isochronous data transfer modes
- 4.35V to 5.25V single supply operating voltage range
- UHCI / OHCI / EHCI host controller compatible
- USB 2.0 Full Speed (12 Mbits / Second) compatible
- Compact 48-LD LQFP package

VIRTUAL COM PORT (VCP) DRIVERS for

- Windows 98 / 98 SE / 2000 / ME / XP
- Linux 2.40 and greater
- Windows CE
- MAC OS-8 and OS-9**
- MAC OS-X**

D2XX (USB Direct Drivers + DLL S/W Interface)

- Windows 98 / 98 SE / 2000 / ME / XP

APPLICATION AREAS

- USB ⇔Dual Port RS232 Converters
- USB ⇔ Dual Port RS422 / RS485 Converters
- Upgrading Legacy Peripheral Designs to USB
- USB Instrumentation
- USB JTAG Programming
- USB to SPI Bus Interfaces
- USB Industrial Control
- Field Upgradable USB Products
- Galvanically Isolated Products with USB Interface

[** = In planning or under development]

3.0 Simplified Block Diagram

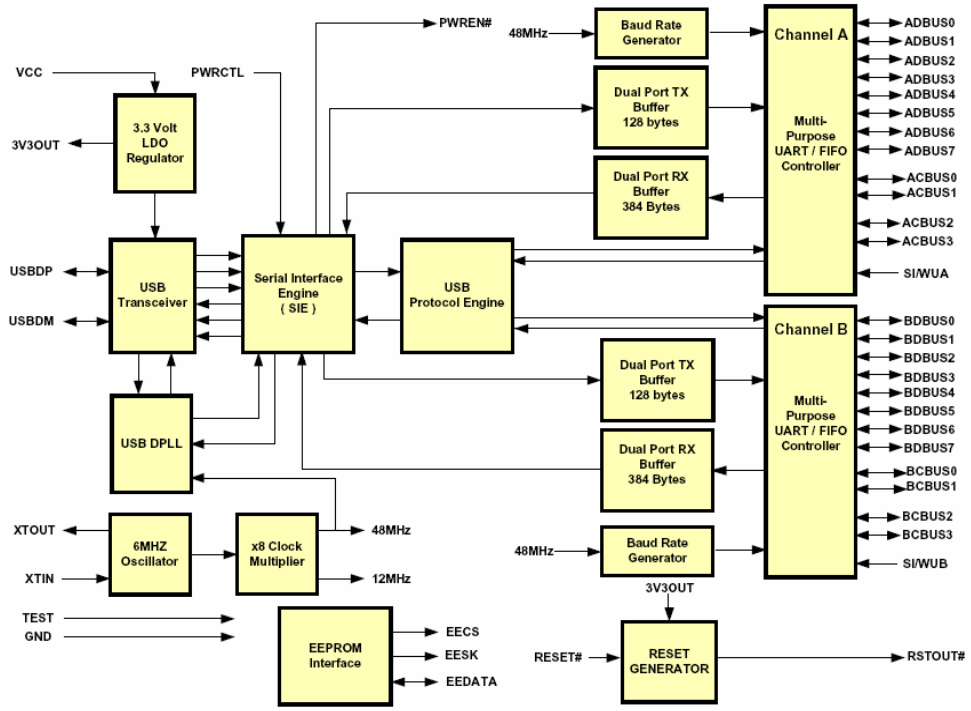


Figure 1 - FT2232C Simplified Block Diagram

5.0 Pin Definitions

This section decribes the operation of the FT2232C pins. Common pins are defined in the first section, then the I/O pins are defined, by chip mode. More detailed descriptions of the operation of the I/O pins are provided in section 9.

5.1 Common Pins

The operation of the following FT2232C pins stay the same, regardless of the chip mode :-

USB INTERFACE GROUP

Pin#	Signal	Type	Description
7	USBDP	I/O	USB Data Signal Plus (Requires 1.5K pull-up to 3V3OUT or RSTOUT#)
8	USBDM	I/O	USB Data Signal Minus

EEPROM INTERFACE GROUP

Pin#	Signal	Type	Description
48	EECS	I/O	EEPROM – Chip Select. Tri-State during device reset. **Note 1
1	EESK	OUTPUT	Clock signal to EEPROM. Tri-State during device reset, else drives out. **Note 1
2	EEDATA	I/O	EEPROM – Data I/O Connect directly to Data-In of the EEPROM and to Data-Out of the EEPROM via a 2.2K resistor. Also, pull Data-Out of the EEPROM to VCC via a 10K resistor for correct operation. Tri-State during device reset. **Note 1

MISCELLANEOUS SIGNAL GROUP

Pin#	Signal	Type	Description
4	RESET#	INPUT	Can be used by an external device to reset the FT2232C. If not required, tie to VCC. **Note 1
5	RSTOUT#	OUTPUT	Output of the internal Reset Generator. Drives low for 5.6 ms after VCC > 3.5V and the internal clock starts up, then clamps it's output to the 3.3V output of the internal regulator. Taking RESET# low will also force RSTOUT# to drive low. RSTOUT# is NOT affected by a USB Bus Reset.
47	TEST	INPUT	Puts device into I.C. test mode – must be tied to GND for normal operation.
41	PWREN#	OUTPUT	Goes Low after the device is configured via USB, then high during USB suspend. Can be used to control power to external logic using a P-Channel Logic Level MOSFET switch. Enable the Interface Pull-Down Option in EEPROM when using the PWREN# pin in this way.
43	XTIN	INPUT	Input to 6MHz Crystal Oscillator Cell. This pin can also be driven by an external 6MHz clock if required. Note : Switching threshold of this pin is VCC/2, so if driving from an external source, the source must be driving at 5V CMOS level or a.c. coupled to centre around VCC/2.
44	XTOUT	OUTPUT	Output from 6MHz Crystal Oscillator Cell. XTOUT stops oscillating during USB suspend, so take care if using this signal to clock external logic.

**Note 1 - *During device reset, these pins are tri-state but pulled up to VCC via internal 200K resistors.*

FT2232C Dual USB UART / FIFO I.C.

POWER AND GND GROUP

Pin#	Signal	Type	Description
6	3V3OUT	OUTPUT	3.3 volt Output from the integrated L.D.O. regulator This pin should be decoupled to GND using a 33nF ceramic capacitor in close proximity to the device pin. It's prime purpose is to provide the internal 3.3V supply to the USB transceiver cell and the RSTOUT# pin. A small amount of current (<= 5mA) can be drawn from this pin to power external 3.3V logic if required.
3, 42	VCC	PWR	+4.35 volt to +5.25 volt VCC to the device core, LDO and non-UART / FIFO controller interface pins.
14	VCCIOA	PWR	+3.0 volt to +5.25 volt VCC to the UART / FIFO A Channel interface pins 10..13, 15..17 and 19..24. When interfacing with 3.3V external logic in a bus powered design connect VCCIO to a 3.3V supply generated from the USB bus. When interfacing with 3.3V external logic in a self powered design connect VCCIO to the 3.3V supply of the external logic. Otherwise connect to VCC to drive out at 5V CMOS level.
31	VCCIOB	PWR	+3.0 volt to +5.25 volt VCC to the UART / FIFO B Channel interface pins 26..30, 32..33 and 35..40. When interfacing with 3.3V external logic in a bus powered design connect VCCIO to a 3.3V supply generated from the USB bus. When interfacing with 3.3V external logic in a self powered design connect VCCIO to the 3.3V supply of the external logic. Otherwise connect to VCC to drive out at 5V CMOS level.
9,18, 25, 34	GND	PWR	Device - Ground Supply Pins
46	AVCC	PWR	Device - Analog Power Supply for the internal x8 clock multiplier. A low pass filter consisting of a 470 Ohm series resistor and a 100 nF to GND should be used on the supply to this pin.
45	AGND	PWR	Device - Analog Ground Supply for the internal x8 clock multiplier

9.3 245 FIFO Interface Mode Signal Descriptions and Configuration

When either Channel A or Channel B are in 245 FIFO mode the IO signal lines are configured as follows.

FIFO DATA BUS GROUP **Note 17

Pin#		Signal	Type	Description
Channel A	Channel B			
24	40	D0	I/O	FIFO Data Bus Bit 0
23	39	D1	I/O	FIFO Data Bus Bit 1
22	38	D2	I/O	FIFO Data Bus Bit 2
21	37	D3	I/O	FIFO Data Bus Bit 3
20	36	D4	I/O	FIFO Data Bus Bit 4
19	35	D5	I/O	FIFO Data Bus Bit 5
17	33	D6	I/O	FIFO Data Bus Bit 6
16	32	D7	I/O	FIFO Data Bus Bit 7

FIFO CONTROL INTERFACE GROUP

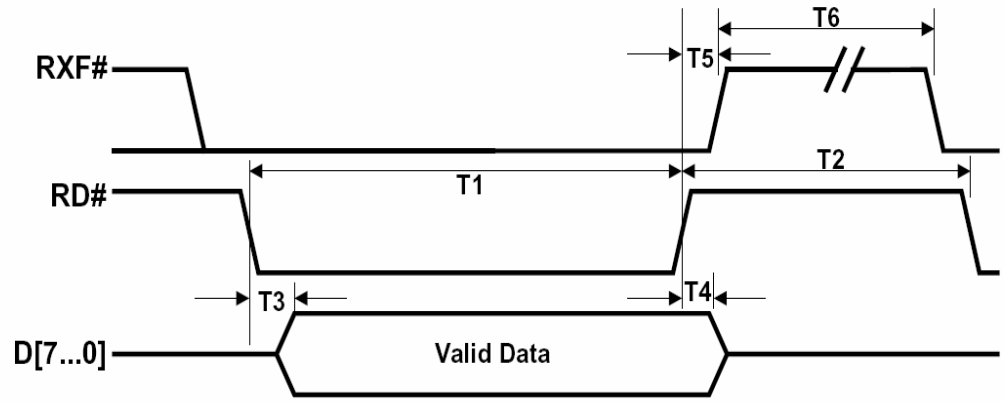
Pin#		Signal	Type	Description
Channel A	Channel B			
15	30	RXF#	OUTPUT	When high, do not read data from the FIFO. When low, there is data available in the FIFO which can be read by strobing RD# low then high again ** Note 18
13	29	TXE#	OUTPUT	When high, do not write data into the FIFO. When low, data can be written into the FIFO by strobing WR high then low. ** Note 18
12	28	RD#	INPUT	Enables Current FIFO Data Byte on D0..D7 when low. Fetches the next FIFO Data Byte (if available) from the Receive FIFO Buffer when RD# goes from low to high. ** Note 17
11	27	WR	INPUT	Writes the Data Byte on the D0..D7 into the Transmit FIFO Buffer when WR goes from high to low. ** Note 17
10	26	SI/WU	INPUT	The Send Immediate / WakeUp signal combines two functions on a single pin. If USB is in suspend mode (PWREN# = 1) and remote wakeup is enabled in the EEPROM , strobing this pin low will cause the device to request a resume on the USB Bus. Normally, this can be used to wake up the Host PC. During normal operation (PWREN# = 0), if this pin is strobed low any data in the device TX buffer will be sent out over USB on the next Bulk-IN request from the drivers regardless of the pending packet size. This can be used to optimise USB transfer speed for some applications. Tie this pin to VCCIO if not used.

****Note 17 :** *In Input Mode, these pins are pulled to VCCIO via internal 200K resistors. These can be programmed to gently pull low during USB suspend (PWREN# = "1") by setting this option in the EEPROM.*

****Note 18 :** *During device reset, these pins are tri-state but pulled up to VCCIO via internal 200K resistors.*

245 FIFO Mode TIMING DIAGRAMS

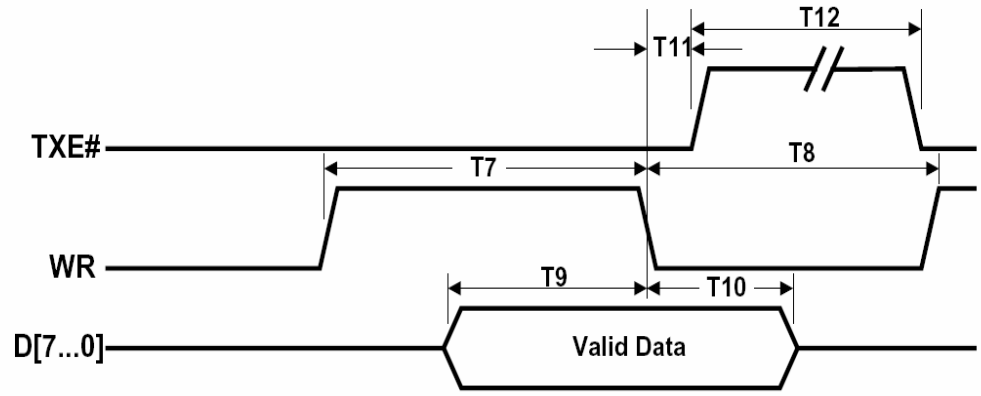
Figure 19 - FIFO READ Cycle



Time	Description	Min	Max	Unit
T1	RD# Active Pulse Width	50		ns
T2	RD# to RD Pre-Charge Time	50 + T6		ns
T3	RD# Active to Valid Data **Note 19	20	50	ns
T4	Valid Data Hold Time from RD# Inactive **Note 19	0		ns
T5	RD# Inactive to RXF#	0	25	ns
T6	RXF# inactive after RD# cycle	80		ns

** Note 19 : *Load 30 pF at standard drive level. These times will also vary if the high output drive level is enabled.*

Figure 20 - FIFO Write Cycle



Time	Description	Min	Max	Unit
T7	WR Active Pulse Width	50		ns
T8	WR to WR Pre-Charge Time	50		ns
T9	Data Setup Time before WR inactive	20		ns
T10	Data Hold Time from WR inactive	0		ns
T11	WR Inactive to TXE#	5	25	ns
T12	TXE inactive after WR cycle	80		ns

APPENDIX B.2

Features

- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-Level Program Memory Lock
- 128 x 8-Bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low Power Idle and Power Down Modes

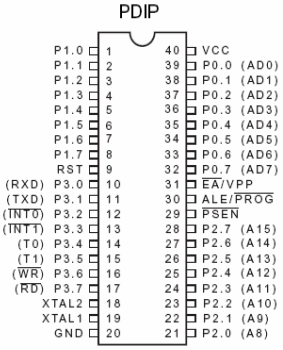
Description

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash Programmable and Erasable Read Only Memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

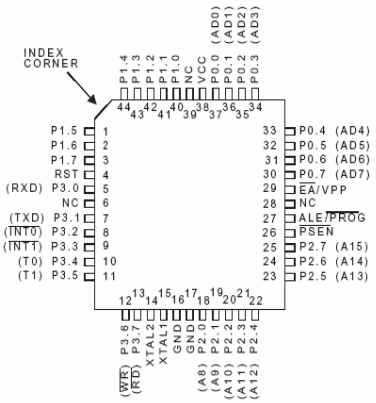
(continued)

Pin Configurations

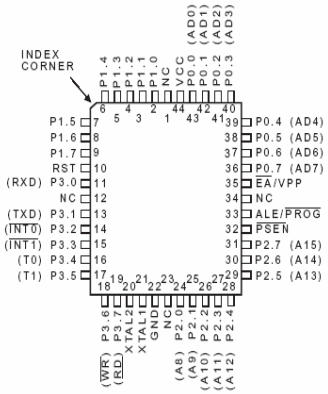
PDIP



PQFP/TQFP



PLCC



8-Bit Microcontroller with 4K Bytes Flash

AT89C51

0265F-A–12/97

The AT89C51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Description

V_{CC}
Supply voltage.

GND
Ground.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1
Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2
Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and verification.

RST
Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$
Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

$\overline{PSEN}$
Program Store Enable is the read strobe to external program memory.

When the AT89C51 is executing code from external program memory, $\overline{PSEN}$ is activated twice each machine cycle, except that two $\overline{PSEN}$ activations are skipped during each access to external data memory.

$\overline{EA}$/V_{PP}
External Access Enable. $\overline{EA}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming, for parts that require 12-volt V_{PP}.

XTAL1
Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2
Output from the inverting oscillator amplifier.

Oscillator Characteristics

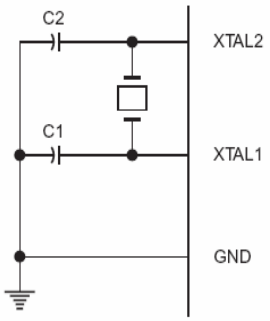
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

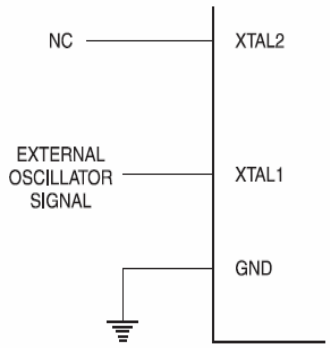
It should be noted that when idle is terminated by a hard ware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

Figure 1. Oscillator Connections



Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Figure 2. External Clock Drive Configuration



Status of External Pins During Idle and Power Down Modes

Mode	Program Memory	ALE	$\overline{PSEN}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power Down	Internal	0	0	Data	Data	Data	Data
Power Down	External	0	0	Float	Data	Data	Data

APPENDIX B.3

Sipex SP21EHCA

National Semiconductor

October 1999

ADC0808/ADC0809
8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE® outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 data sheet. (See AN-247 for more information.)

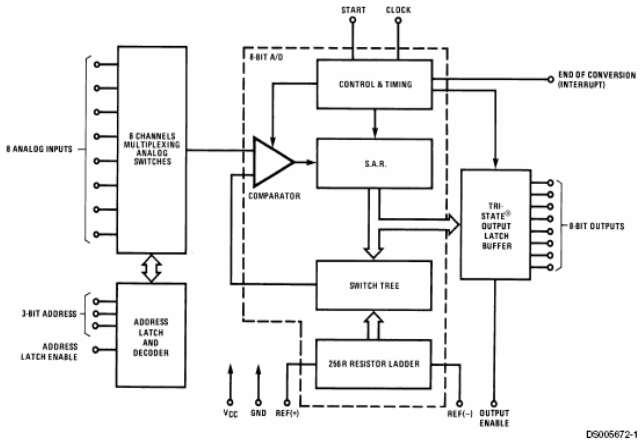
Features

- Easy interface to all microprocessors
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- No zero or full-scale adjust required
- 8-channel multiplexer with address logic
- 0V to 5V input range with single 5V power supply
- Outputs meet TTL voltage level specifications
- Standard hermetic or molded 28-pin DIP package
- 28-pin molded chip carrier package
- ADC0808 equivalent to MM74C949
- ADC0809 equivalent to MM74C949-1

Key Specifications

■ Resolution	8 Bits
■ Total Unadjusted Error	±½ LSB and ±1 LSB
■ Single Supply	5 V_{DC}
■ Low Power	15 mW
■ Conversion Time	100 µs

Block Diagram



See Ordering Information

TRI-STATE® is a registered trademark of National Semiconductor Corp.

Functional Description

Multiplexer. The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. *Table 1* shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

TABLE 1.

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

The Converter

The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach (*Figure 1*) was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in *Figure 1* are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached +½ LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. *Figure 2* shows a typical example of a 3-bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bits using the 256R network.

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed through a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors.

Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

APPENDIX

C

PERIPHERAL FIRMWARE CODE

```
;=====Equates for port bits interfaced to ADC0808's========

ADC_DATA        EQU   P2              ;digital data in port

ADC_OE          BIT   P3.7            ;output enable for ADC
ADC_SC          BIT   P3.4            ;start conversion
ADC_EOC         BIT  P3.6             ;end of conversion
ADC_ALE         BIT   P3.5            ;address latch enable
ADC_CLK         BIT  P1.7             ;clock signal for ADC

AD0             BIT   P1.4            ;multiplexer channel
AD1             BIT   P1.5            ;addressing
AD2             BIT   P1.6

;=====Equates for port bits interfaced to DLP2232M Module==

USB_DATA        EQU   P0              ;usb data out port

RXF             BIT   P1.0            ;receive status signal
TXE             BIT   P1.1            ;transmit status signal
READ            BIT   P1.2            ;read control signal
WRITE           BIT   P1.3            ;write control signal

;=====Main program indefinite loop=======================

                ORG 0
                jmp MAIN

MAIN:           setb RXF              ;initial state for
                setb TXE              ;DLP2232M module
                setb READ             ;interface signals
                clr WRITE

                clr ADC_OE
                clr ADC_ALE
                clr ADC_SC
                clr ADC_CLK
                mov r0,#07h           ;channel counter
                mov r1,#50h           ;destination pointer
                mov r2,#00h           ;channel selection
```

```
CONVERSION:     mov a,r2
                orl p1,a
                call START_CONV       ;A/D conversion routine
                inc r1                ;refresh destination
                                      ;pointer
                mov a,#10h            ;next channel address
                add a,r2              ;selection
                mov r2,a
                djnz r0,CONVERSION

                mov r0,#07h           ;channel counter
                mov r1,#50h           ;source pointer
TRANSMIT:       call TRANSMIT_DATA    ;data transmit routine
                mov a,#10h            ;next channel address
                add a,r2              ;selection
                mov r2,a
                djnz r0,TRANSMIT
                jmp MAIN


;=====Subroutine for A to D conversion using ADC===========

START_CONV:     mov ADC_DATA,#0ffh

                clr ADC_OE            ; hi Z output
                setb ADC_EOC
                setb ADC_ALE          ;start conversion
                setb ADC_SC

                clr ADC_ALE
                clr ADC_SC

START_CLK:      cpl ADC_CLK
                jnb ADC_EOC,START_CLK
                setb ADC_OE
                mov @r1,ADC_DATA
                clr ADC_OE
                ret

;=====Subroutine for transmitting data to DLP2232M Module==

TRANSMIT_DATA:  setb TXE
HERE:           jb TXE, HERE
                setb WR
                mov USB_DATA,@r1
                nop
                clr WR
                ret

END
```

APPENDIX

D

HOST COMPUTER SOFTWARE

D.1 Search Button Function for searching the device

```
void CUSBDemoDlg::OnButtonSearch()
{
Search for Descriptions or Serial Numbers depending on the current mode
        FT_STATUS ftStatus;
        DWORD numDevs;

        Close(); //must be closed to perform the ListDevices() function
        UpdateData(TRUE);
        m_PortStatus = ("DLP-USB1 Closed.");
        m_Search = ("");
        m_Received.ResetContent();
        UpdateData(FALSE);
        UpdateWindow();

        ftStatus = ListDevices(&numDevs, NULL, FT_LIST_NUMBER_ONLY);
        if(ftStatus == FT_OK)
        {
 FT_ListDevices OK, Show number of devices connected in list box
                CString str;
                str.Format("%d device(s) attached:", (int)numDevs);
                m_Received.AddString(str);

If current mode is open "by device #" then list device numbers
                if((m_SerDescr==2) && (numDevs>0))
```

```
{for(DWORD d=0; d<numDevs; d++)
        {
                str.Format("%d", d);
                m_Received.AddString(str);
        }}
```

If current mode is open "by description" then list descriptions of all connected devices

```
if((m_SerDescr==0) && (numDevs>0))
{
ftStatus = ListDevices(&numDevs, NULL, FT_LIST_NUMBER_ONLY);
        if(ftStatus == FT_OK)
        {
                char *BufPtrs[64]; // pointer to array of 64 pointers
                for(DWORD d=0; d<numDevs; d++)
                BufPtrs[d] = new char[64];
                BufPtrs[d] = NULL;
                ftStatus=ListDevices(BufPtrs,&numDevs, FT_LIST_ALL|FT_OPEN_BY_DESCRIPTION);
                if (FT_SUCCESS(ftStatus))
                {
                        for(DWORD u=0; u<numDevs; u++)
                        {
                                str.Format("%s", BufPtrs[u]);
                                m_Received.AddString(str);
                        }
                }
                else
                {
                        str.Format("ListDevices failed");
                        m_Received.AddString(str);
                }
```

```
}
                                        }
```

If current mode is open "by serial number" the list descriptions of all connected devices

```
        if((m_SerDescr==1) && (numDevs>0))
        {
        ftStatus = ListDevices(&numDevs, NULL, FT_LIST_NUMBER_ONLY);
                if(ftStatus == FT_OK)
                {
                        char *BufPtrs[64]; // pointer to array of 64 pointers
                        for(DWORD d=0; d<numDevs; d++)
                                BufPtrs[d] = new char[64];
                        BufPtrs[d] = NULL;

                        ftStatus = ListDevices(BufPtrs, &numDevs, FT_LIST_ALL|FT_OPEN_BY_SERIAL_NUMBER);
                        if (FT_SUCCESS(ftStatus))
                        {
                                for(DWORD u=0; u<numDevs; u++)
                                {
                                        str.Format("%s", BufPtrs[u]);
                                        m_Received.AddString(str);
                                }
                        }
                        else
                        {
                                str.Format("ListDevices failed");
                                m_Received.AddString(str);
                        }
}
```

```
}
        else
        {
 FT_ListDevices failed
                AfxMessageBox("FT_ListDevices failed");
        }
}
```

D.2 Open Button Function for Opening the Device

```
void CUSBDemoDlg::OnButtonOpen()
{
        unsigned char txbuf[25], rxbuf[25];
        DWORD ret_bytes;
        UpdateData(TRUE);
        m_PortStatus = ("Reset");
        UpdateData(FALSE);
        UpdateWindow();
        Close();

Open the device
        FT_STATUS status = OpenBy();
        if(status>0)
        {
                m_PortStatus = ("Could not open DLP-USB1");
                board_present=0;
        }
        else
        {
                ResetDevice();
                Purge(FT_PURGE_RX || FT_PURGE_TX);
                ResetDevice();
```

Extend timeout while board finishes reset

```
SetTimeouts(3000, 3000);
```

Test for presence of board

```
txbuf[0] = 0x80;
Write(txbuf, 8, &ret_bytes);
Read(rxbuf, 8, &ret_bytes);
if(ret_bytes==0)
        Read(rxbuf, 1, &ret_bytes);
if(rxbuf[0] != 0x80)
{
        m_PortStatus = ("DLP-USB2 not responding");
        int c = (int)rxbuf[0];
        CString str;
        str.Format(" (0x%.2X)", c);
        board_present=0;
        Close();
}
else
{
        m_PortStatus = ("DLP-USB2 ready.");
        board_present=1;
```

Data to be split into various channels

```
        int c =(int)rxbuf[0];
        CString str;
        str.Format("(0x%.2X)",c);
        m_Ch0 = str;

        int c1 =(int)rxbuf[1];
        CString str1;
        str1.Format("(0x%.2X)",c1);
        m_Ch1 = str1;
```

```
            int c2 =(int)rxbuf[2];
            CString str2;
            str2.Format("(0x%.2X)",c2);
            m_Ch2 = str2;

            int c3 =(int)rxbuf[3];
            CString str3;
            str3.Format("(0x%.2X)",c3);
            m_Ch3 = str3;

            int c4 =(int)rxbuf[4];
            CString str4;
            str4.Format("(0x%.2X)",c4);
            m_Ch4 = str4;

            int c5 =(int)rxbuf[5];
            CString str5;
            str5.Format("(0x%.2X)",c5);
            m_Ch5 = str5;

            int c6 =(int)rxbuf[6];
            CString str6;
            str6.Format("(0x%.2X)",c6);
            m_Ch6 = str6;

            int c7 =(int)rxbuf[7];
            CString str7;
            str7.Format("(0x%.2X)",c7);
            m_Ch7 = str7;
        }
    }
```

```
	SetTimeouts(300, 300);
	UpdateData(FALSE);
	UpdateWindow();
}
FT_STATUS CUSBDemoDlg::OpenBy()
{
	UpdateData(TRUE);
	if(m_Search.Find("Enter device Description") > -1)
	{
```

Highlight the descriptor/serial number to draw attention...

```
		CEdit *pEdt;
		pEdt = (CEdit *)GetDlgItem(IDC_EDIT_SEARCH);
		pEdt->SetFocus();
		pEdt->SetSel(0, -1, FALSE);
		AfxMessageBox("You must enter a Device Description or Serial Number.");
		return FT_DEVICE_NOT_OPENED;
	}
	FT_STATUS status;
	ULONG x=0;
	if(m_SerDescr==0)
		status = OpenEx((PVOID)(LPCTSTR)m_Search, FT_OPEN_BY_DESCRIPTION);
	if(m_SerDescr==1)
		status = OpenEx((PVOID)(LPCTSTR)m_Search, FT_OPEN_BY_SERIAL_NUMBER);
```

If open by device OR no method was selected

```
	if((m_SerDescr==2) || (m_SerDescr==-1))	{
```

Nothing entered - open default device 0

```
		if(m_Search.GetLength() < 1)
		{
```

Load default device 0

```
                    status = Open((PVOID)x);            }
        else
        {
                    if(m_Search.GetLength() > 2)                    {
AfxMessageBox("Select a method to open or enter a valid device number (0-64).");
                        return FT_DEVICE_NOT_OPENED;
                    }
```

Convert string to decimal number and open(x)

```
                    char str[3];
                    strcpy(str, (LPCTSTR)m_Search);
                    x = atoi(str);
                    if((x<0) || (x>64))
                    {
AfxMessageBox("Select a method to open or enter a valid device number (0-64).");
                        return FT_DEVICE_NOT_OPENED;
                    }
                    status = Open((PVOID)x);
        }
    }
    return status;
}
```

D.3 Clear Button Function for clearing the contents of list box and channel edit boxes

```
void CUSBDemoDlg::OnButtonClear()
{
        UpdateData(TRUE);
        m_Ch0 =" ";
        m_Ch1= " ";
```

```
        m_Ch2 =" ";
        m_Ch3= " ";
        m_Ch4 =" ";
        m_Ch5= " ";
        m_Ch6 =" ";
        m_Ch7= " ";
        m_Received.ResetContent();
        UpdateData(FALSE);
        UpdateWindow();
}
```

D.4 Exit Button Function

```
void CUSBDemoDlg::OnButtonExit()
{
    OnOK();
}
```

D.5 Description Radio Button Function

```
void CUSBDemoDlg::OnRadioDescription()
{
    CWinApp* pApp = AfxGetApp();
    pApp->WriteProfileInt("MyUSBTestAp", "SerDesc", 0);
    UpdateData(TRUE);
    m_Search = AfxGetApp()->GetProfileString("MyUSBTestAp", "DescString",
"Enter device Description or Serial Number here.");
    UpdateData(FALSE);
    UpdateWindow();
}
```

D.6 Serial Number Radio Button Function

```
void CUSBDemoDlg::OnRadioSernum()
{
        CWinApp* pApp = AfxGetApp();
        pApp->WriteProfileInt("MyUSBTestAp", "SerDesc", 1);
        UpdateData(TRUE);
        m_Search = AfxGetApp()->GetProfileString("MyUSBTestAp", "SerialString",
"Enter device Description or Serial Number here.");
        UpdateData(FALSE);
        UpdateWindow();
}
```

D.7 Device Number Radio Button Function

```
void CUSBDemoDlg::OnRadioDevno()
{
        CWinApp* pApp = AfxGetApp();
        pApp->WriteProfileInt("MyUSBTestAp", "SerDesc", 2);
        UpdateData(TRUE);
        m_Search = AfxGetApp()->GetProfileString("MyUSBTestAp", "DevNmbr",
"Enter device Description or Serial Number here.");
        UpdateData(FALSE);
        UpdateWindow();
}
```

List of references:

Books:

- Axelson J. *USB Complete,* 2nd ed. Penram International Publishing (India).
- Hyde J. *USB Design by Example,* 2nd ed. Intel Press.
- Ayala, K. J. *The 8051 Microcontroller Architecture, Programming and Applications.* 2nd ed. Penram International Publishing (India).
- Predko, M. *Programming and customizing the 8051 microcontroller.* 1st ed. Tata McGraw-Hill Publishing Company.

Websites:

- www.usb.org The USB Implementers Forum, Inc. (USB-IF)
- www.usbdeveloper.com/index.htm
- www.design-by-example.com
- www.epanorama.net/links/pc/index.html
- www.usbman.com/developer.htm
- www.Lvr.com
- www.ftdichip.com
- www.tifr.res.in
- www.dlpdesign.com/usb/
- www.cypress.com
- www.atmel.com
- www.semiconductors.philips.com
- www.nationalsemiconductor.com
- www.beyondlogic.org/usbnutshell/usb7.htm
- www.embedded.com/2000/0003/0003ia2.htm
- www.stmicroelectonics.com
- www.vesit.edu