EE 318 Electronic Design Lab -I, Project Report, EE Dept., IIT Bombay, March 2007

Online Data Acquisition and Transfer to PC through USB

Group No. D4
Aman Jain (04d07009) <amanjain@ee.iitb.ac.in>
Rahul Singh Solanki (04d07007) <rahulss@ee.iitb.ac.in>
Aseem Manmualiya (04d07031) <aseem@ee.iitb.ac.in>

Supervisor: Prof. H. Narayanan
Under the guidance of Prof. S. V. Kulkarni

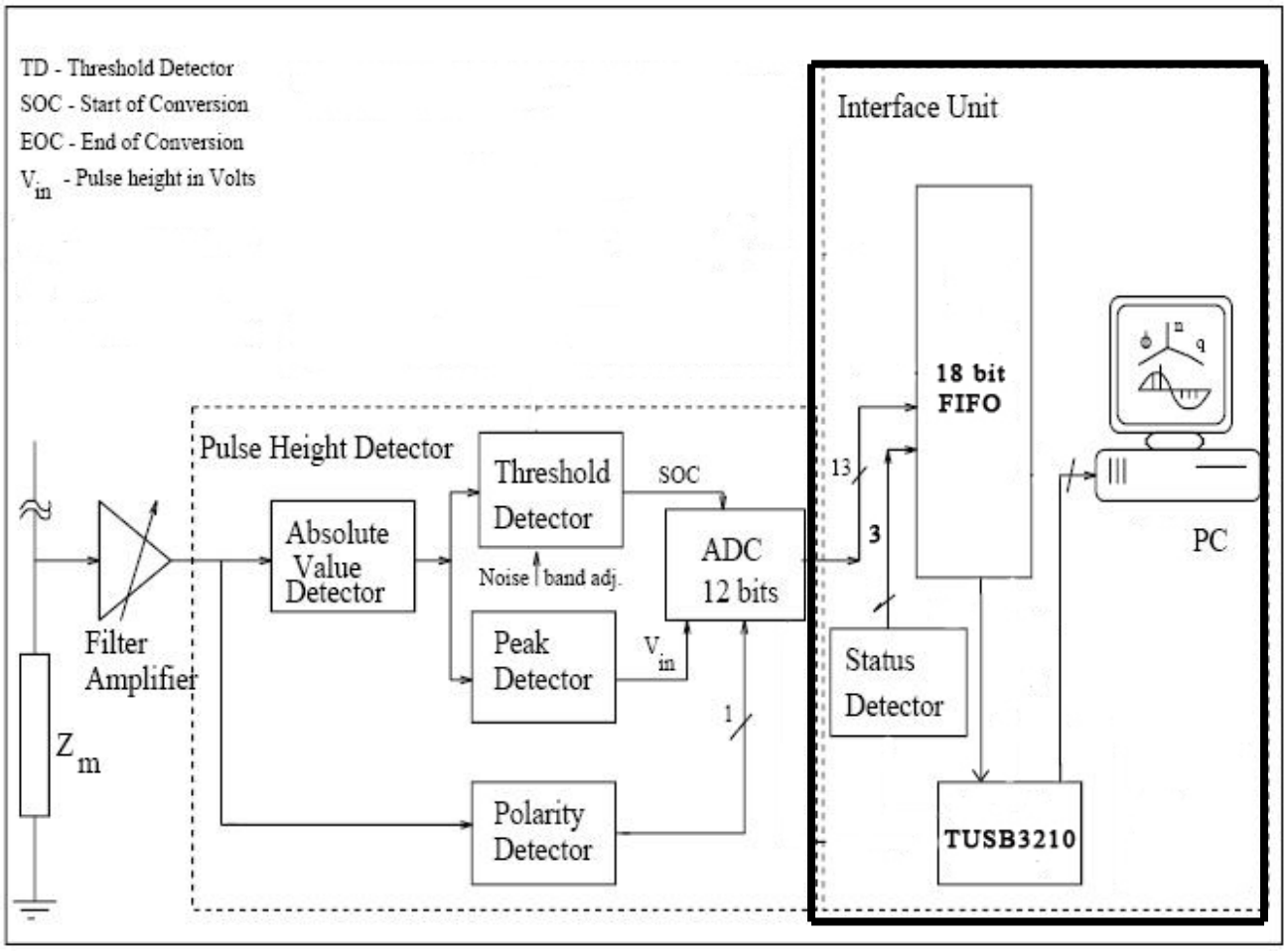
Abstract: The objective of this design project is to build a TUSB3210 based system in which the data from a very fast ADC is to be fed into a host-PC via a USB interface. The data rate is very high (~10-17 MSPS) at 16 bit resolution, the data rate being a variable entity based on the stochastically distributed partial discharges occurring inside high voltage equipments' insulators), and hence we implement a dual port FIFO so as to store data from ADC and transfer it to the host-PC almost simultaneously, with the minimal data loss possible which is inevitable because the host might not always be in a position to accept data owing to it's OS constraints and system interrupts.

Introduction: The PRPHA (Pulse Resolved Pulse Height Detector) essentially has a pulse height detector and an interface unit to PC. The function of the pulse height detector is to identify the pulse, sample the peak of the pulse (height) and digitize it. The interface unit synchronizes the height and phase data transfer to the PC. The PC based phase resolved detectors measure the PD pulses sequentially and make a record of it on the computer disk but have limitations on data acquisition speed, processing and memory utilization. The present work is aimed at developing an indigenous technology for interfacing partial discharge measurements so as to analyze them with ease. Speed and data loss, along with ergonomics are the considerations to be dealt with.

Design Approach: The block diagram consists of :

A) Pulse Height Detector
  i) Absolute Value Detector (12 bits)
  ii) Polarity Detector (1 bit)
B) Status Detectors (3 bits)
C) Interface Unit (FIFO+TUSB synchronization)

This is precisely what our contribution to the project is, that is, the highlighted portion in the block diagram underneath.



A) The Pulse Height Detector transmits a 12- bit resolution absolute value detector data, and a 1-bit polarity bit detector data. Along with this, the ADC used to digitize the Pulse Height transmits an EOC(End Of Conversion) Signal.
B)Status detectors: Three status bits are included in the data to the interface unit as shown in the figure underneath.

Pulse event	Data type	Pulse data (binary)	Pulse data range (hex)
Positive PD pulse	Amplitude	0 0 0 0 aaaa aaaa aaaa	0000 - 0fff
Negative PD Pulse	Amplitude	0 0 0 1 aaaa aaaa aaaa	1000 - 1fff
Marker pulse	Amplitude	Last PD pulse amplitude	0000 - 1fff
PD pulse with marker	Amplitude	0 1 1 P aaaa aaaa aaaa	6000 - 7fff

PD record details with status bits

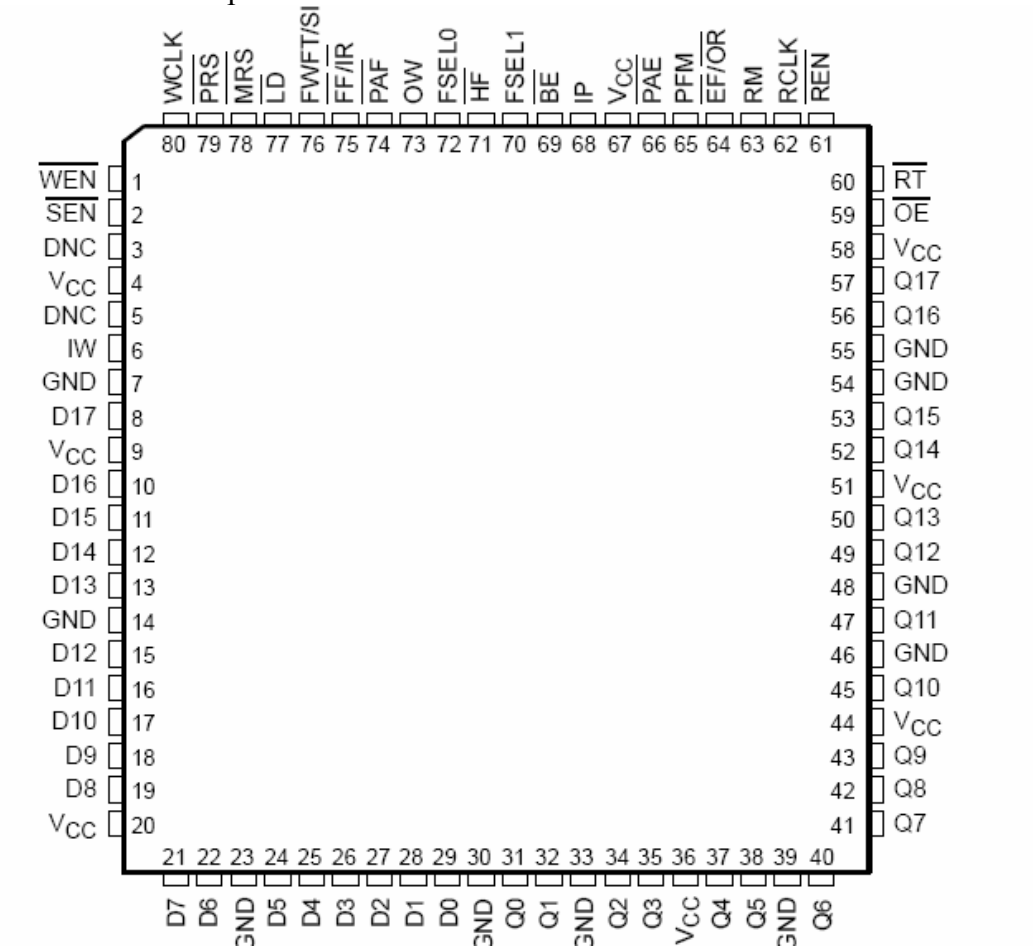
C) Interface Unit: The Circuit which links the detectors and the PC, so that they can converse with proper protocols, and data can be saved in the host-PC with proper synchronization for later analysis of the PD data, is the "Interface Unit". The interface unit is designed to give the optimum usage of the USB Port. It is not possible to directly take data to the PC without accumulating it anywhere, for even the fastest PC's may not be able to cope up with the instantaneous attentions required by the data, for the reasons lying in Operating System and BIOS. Hence we use a FIFO (First-In-First-Out) Interface, to give temporary storage to the PD Data if the PC cannot cope up with the detectors' speeds.

a)We implement our design with a TI made FIFO which offers us the following features:

FIFO: The SN74V293's are exceptionally deep, high-speed, CMOS first-in first-out (FIFO) memories with clocked read and write controls and a flexible bus-matching x9/x18 data flow. There is flexible x9/x18 bus matching on both read and write ports. This is an absolute blessing for us because of our usage of an 8-bit microcontroller peripheral ahead, which could have accepted only 8-bit data, whereas the data arriving in the FIFO was one with 16-bits' resolution.

The period required by the retransmit operation is fixed and short. The first-word data-latency period, from the time the first word is written to an empty FIFO to the time it can be read, is fixed and short. These FIFO's are particularly appropriate for applications that need to buffer large amounts of data and match buses of unequal sizes. Each FIFO has a data input port (D_n) and a data output port (Q_n), both of which can assume either an 18-bit or 9-bit width, as determined by the state of external control pins' input width (IW)

and output width (OW) during the master-reset cycle. The input port is controlled by write-clock (WCLK) and write-enable (WEN) inputs. Data is written into the FIFO on every rising edge of WCLK when WEN is asserted. The output port is controlled by read-clock (RCLK) and read-enable (REN) inputs. Data is read from the FIFO on every rising edge of RCLK when REN is asserted. An output-enable (OE) input is provided for 3-state control of the outputs.

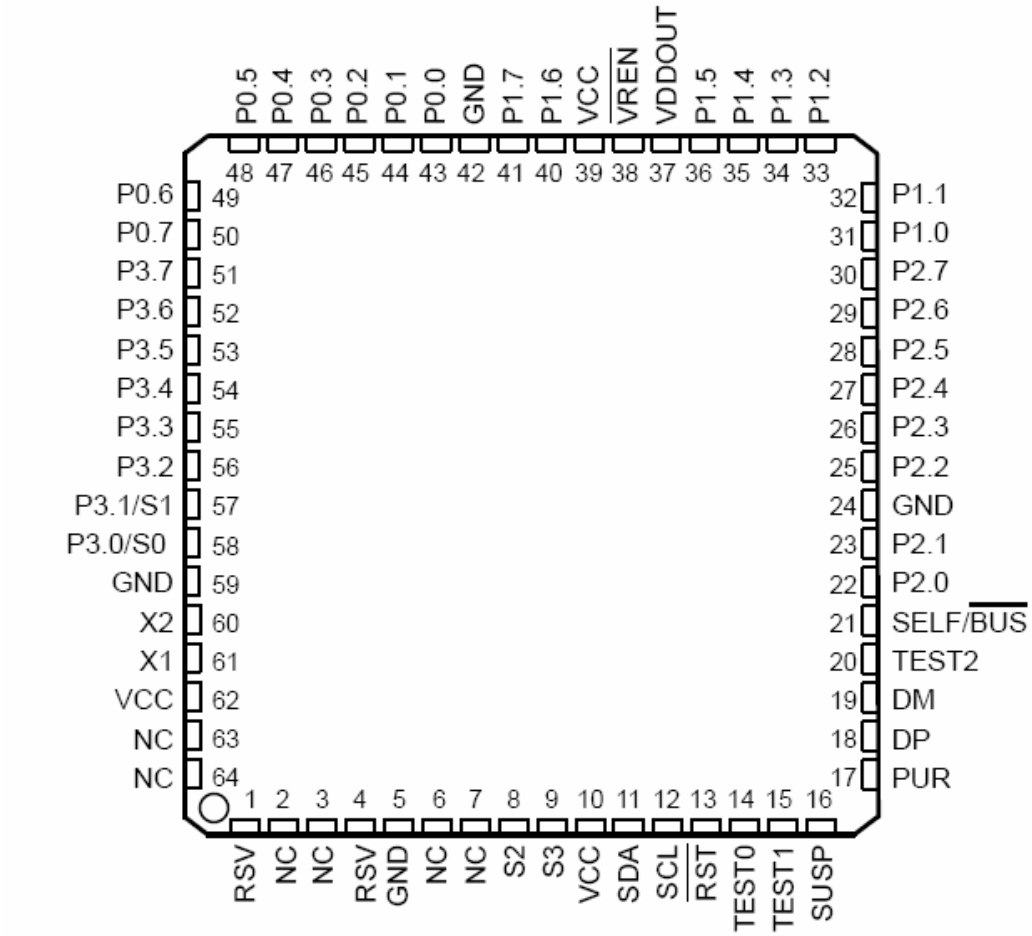


The frequencies of both the RCLK and the WCLK signals can vary from 0 to fMAX, with complete independence. There are no restrictions on the frequency of one clock input with respect to the other. There are two possible timing modes of operation with these devices: first-word fall-through (FWFT) mode and standard mode. We require the standard mode of operation for our purpose. In standard mode, the first word written to an empty FIFO does not appear on the data output lines unless a specific read operation is performed. A read operation, which consists of activating REN and enabling a rising RCLK edge, shifts the word from internal memory to the data output lines.

There are several other features associated with this FIFO, which are not required by us though, like offset programming, we need only Empty Buffer Flag(EF), and Full Buffer Flag(FF).

b) The USB-Interface: The USB 1.1 has several advantages, which make it appropriate a channel for us to transmit PD Data into the PC. We make use of TI made TUSB3210, a Universal Serial Bus General Purpose Device Controller.

TUSB3210: It has a 8k x 8 RAM code space available for downloadable firmware from host or I^2C port and 512 x8 shared RAM used for data buffers and endpoint descriptor blocks (EDB). This is the buffer space for USB packet transactions. In addition, the programmability of the TUSB3210 makes it flexible enough to use for various other general USB I/O applications. Unique vendor identification and product identification (VID/PID) may be selected without the use of an external EEPROM. Using a 12 MHz crystal, the onboard oscillator generates the internal system clocks. The device may be programmed via an inter-IC (I2C) serial interface at power on from an EEPROM, or optionally, the application firmware may be downloaded from a host PC via USB. The popular 8052-based microprocessor allows several third party standard tools to be used for application development. In addition, the vast amounts of application code available in the general market may also be utilized (this may or may not require some code modification due to hardware variations).dependence.



TUSB3210 Boot Operation:
Since the code-space is in RAM (with the exception of the boot ROM), the TUSB3210 firmware must be loaded from an external source. Two options for booting are available:

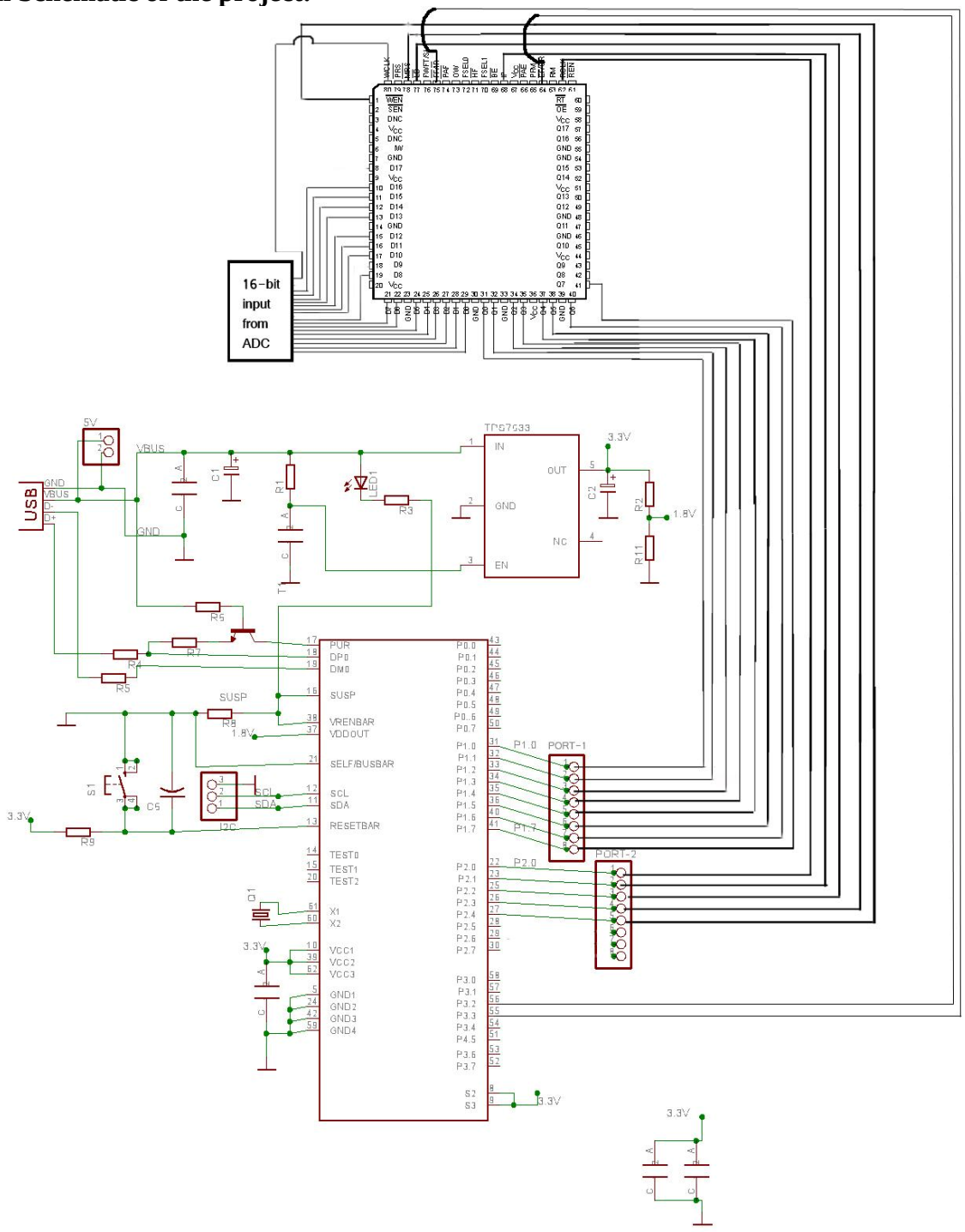
an external serial EEPROM source connected to the I^2C bus, or the host may be used via the USB. On device reset, the SDW bit (in ROM register) and CONT bit in USB Control Register (USBCTL) will be cleared. This will configure the memory space to boot mode (see memory map) and will keep the device *disconnected* from the host. The first instruction will be fetched from location 0000 (which is in the 6k-ROM). The 8k-RAM will be mapped to XDATA space (location 0000h). MCU will execute a read from an external EEPROM and test to see if it contains the code (test for boot signature). If it contains the code, MCU will read from EEPROM and write to the 8k-RAM in XDATA space. If not, MCU will proceed to boot from USB. Once the code is loaded, the MCU will set SDW to 1. This will switch the memory map to normal mode, i.e. the 8k-RAM will be mapped to code space, and the MCU will start executing from location 0000h. Once the switch is done, the MCU will set CONT to 1 (in USBCTL register) This will *connect* the device to the USB bus, resulting in the normal USB device enumeration.

Buffers and I/O RAM:
The address range from FD80 to FFFF is reserved for data buffers, setup packet, endpoint descriptor blocks (EDB), and all I/O. RAM space of 512 bytes [FD80–FF7F] is used for EDB and buffers. The FF80–FFFF range is used for memory mapped registers (MMR). Table 2–1 represents the internal XDATA space allocation.

Endpoint Descriptor Block (EDB-1 to EDB-3):
Data transfers between USB, MCU and external devices are defined by an endpoint descriptor block (EDB). Four input and four output EDBs are provided. With the exception of EDB–0 (I/O Endpoint–0), all EDBs are located in SRAM as shown in Table 2–3. Each EDB contains information describing the X and Y buffers. In addition, it provides general status information.

c) **Voltage Regulation:** We use LM7805 to regulate voltage, with a simple circuitry, so as to provide us 3.3 V supplies needed for our peripherals.

Final Schematic of the project:



Software Development:

We have been able to interface a TUSB PCB with the host computer on which *USBVIEW*, *LIBUSB*, and *SDCC* were required to install for various requirements. The host can, at this stage, converse with the TUSB through a USB interface, send a string, and receive it back. In implementing this, we required uploading a *firmware* (which is written to the 8052 core in the TUSB) into the TUSB through a *bootloader* running on the host, and later on running a *host program* to communicate with the TUSB. The next stage would typically involve writing a firmware to accept data from the FIFO and send it to the host-PC, in which there is still scope for some work. The host program has been written with the task of accepting data from TUSB and writing it to a file on the hard disk.

Hardware development:

We have successfully developed a PCB for the FIFO which takes input from ADC, and sends it to the TUSB, with simultaneous and independent read and write cycles. This PCB is connected with another PCB with a TUSB assembled with proper hardware protocols like reset circuitry, USB type-B connector, and the voltage regulator.