Universal Serial Bus (USB)

USB ACRONYMS

USB – "Universal Serial Bus"
OTG – "On The Go" – similar concept to "Plug and Play" – allows a host to change roles and act as a device
LS- - "Low Speed" – 1.5MB/s (v1.0) – (eg. joystick) – Low BW
FS – "Full Speed" – 12 Mb/s (v1.0) – (eg. disk drive) – Full BW
HS – "High Speed" - 480 Mb/s (v2.0)
SS – "Super Speed" – 5Gb/s (v3.0)
USB-IF – USB Implementers Forum (dev. 2.0 spec – HP, Intel, Lucent, NEC, Philips)
UTMI – USB 2.0 Transceiver Macrocell Interface – standard developed by Intel
UTMI+ - extension to UTMI incl support for OTG
ULPI – (UTMI+ Low Pin Interface) – standard developed for low-pin count (12) discrete USB chip using the UTMI+ interface
– complete i/f also requires front-end PHY circuit

USB System

- Network of Attachments in Logical Star-like Structure and Physical Tree-like Structure with the Host at the Center/Root

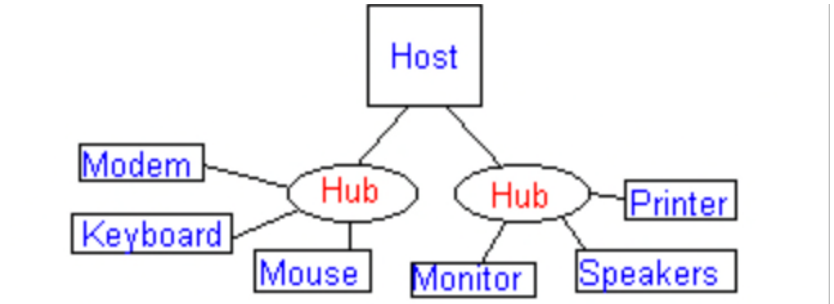


Figure 1(a) : The physical USB arrangement
Functions are joing to hubs in a star arrangement

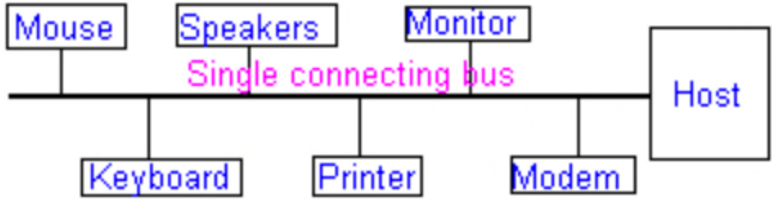


Figure 1(b) : How the USB system appears to functions

http://www.geoffknagge.com/uni/elec101/essay.shtml

USB Terms in Standard

- **Attachments** are **Functions** or **Hubs**
- **Functions** – Peripheral devices like SS disk, mice, keyboards
- **Hubs** – Converts one **Port** to Several Ports
- Hubs and Functions are both called **Devices**

Photo of a DLink USB Hub

http://www.bhphotovideo.com/bnh/controller/home?sku=403654&Q=&O=&is=REG&A=details

USB Function

- Logically Communication between Device and Host Appears to be Traffic over a Single **Bus**
- A Bus is a Single Set of Wires Interconnecting a “Talking” Device and a “Listening” Device
- Later Versions of USB (2.0+) Enable Host to Keep Track of Which Attachments are Present by Sensing When they are Plugged-in to a USB Socket (plug-and-play)

USB Host

- Host is the Center of the Star and Contains a Hub embedded within it – the **Root Hub**
  - Example is a Notebook PC that serves as a USB host/root hub
  - Notebook PC Root Hub has Connections to External USB Connectors (sockets) and possibly Dedicated Internal Devices
- Host keeps Track of Attached Devices by Giving them Unique Numbers (called an **Address**) when it Detects them as Attached

USB Host (cont)

- A Given Device may have a Different Address assigned to it each time it is Attached
- Devices Contain Different Internal Sources/ Destinations for Data Called **Endpoints**
- Endpoints are Either Transmit or Receive Data not both
  - EXAMPLE: keyboard keypad has output endpoint “1” and caps lock light has receiving endpoint “1”
  - Each Device has 16 Possible Endpoints
  - Endpoint Zero Reserved for Configure/control/auto-detect

USB Pipe

- Combination of Address, Endpoint Number, Data Direction (rec or xmit) Defines a **Pipe**
- **Pipe** is Data Path Between Endpoint and Controlling Software
- Special Pipe Contains Endpoint 0: **Default Control Pipe**

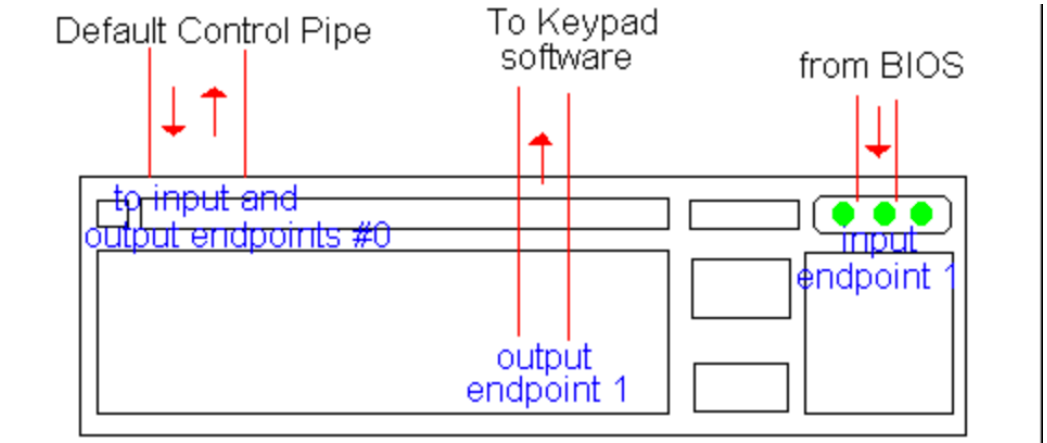


Figure 2 : Simple keyboard model illustrating endpoints and pipes

http://www.geoffknagge.com/uni/elec101/essay.shtml

USB Serial Data Transfer

1) Control Transfer – intended for configuring, controlling, checking status of USB device. Host sends status request to device, later device sends status back
2) Isochronous Transfers – Accuracy not critical, but timing is, for example an audio stream where one garbled frame is unnoticeable (1023 bytes per frame)

USB Serial Data Transfer

3) Interrupt Transfer – small infrequent transfers that require priority over other requests
4) Bulk Transfers – purpose is for transmitting large amounts of data – lowest priority. Useful for things like scanner data

USB Serial Data Transfer

- Serial Data Transfer Means One Bit at a Time

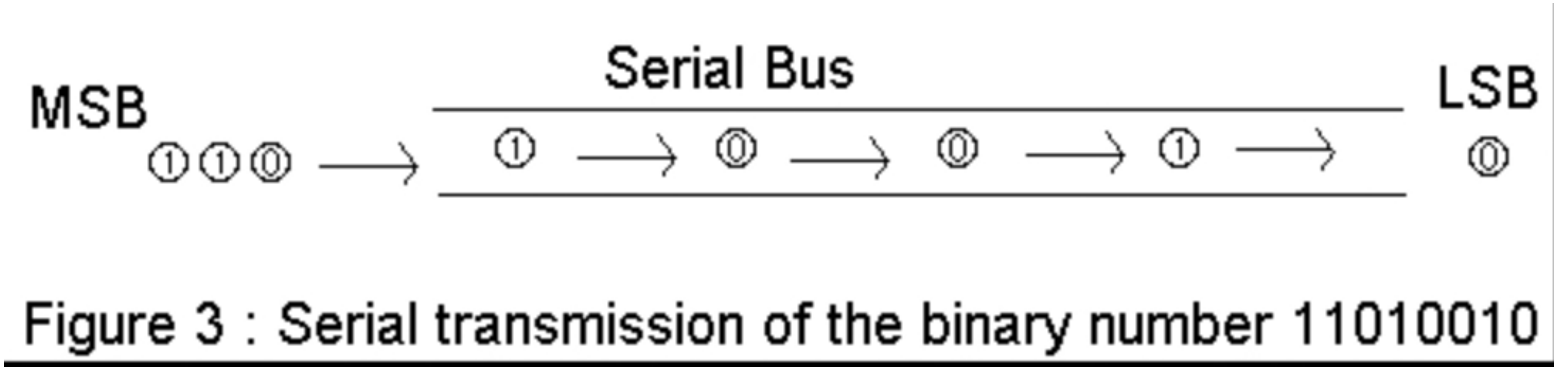


Figure 3 : Serial transmission of the binary number 11010010

http://www.geoffknagge.com/uni/elec101/essay.shtml

- Data Transfer Occurs After Software Sends **I/O Request Packet** (**IRP**) to Appropriate Pipe
- Data Sent in Bundles Called Packets

USB Packets

- USB Data Packet

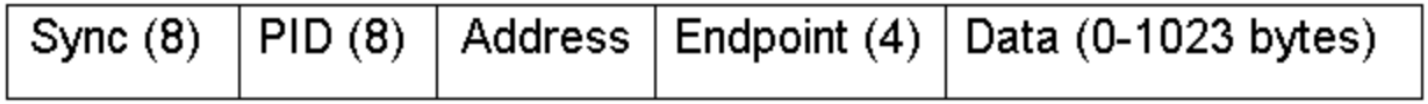


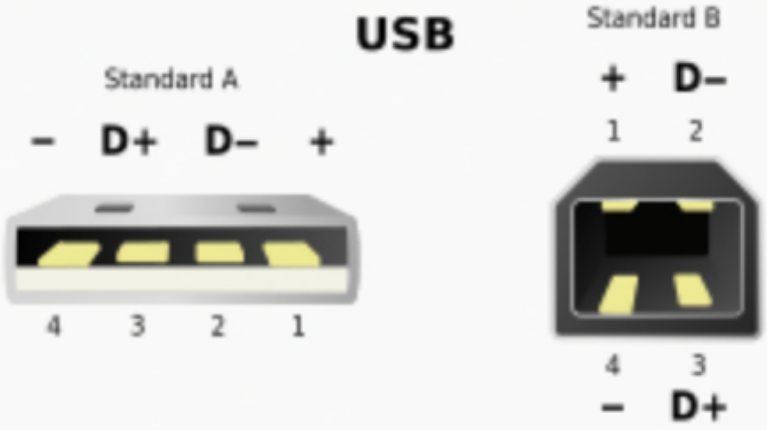
Figure 4 : A typical data packet. Numbers represent size of field in bits, unless otherwise indicated.

http://www.geoffknagge.com/uni/elec101/essay.shtml

- Sync – Used for Timing
- PID – Type/format of data
- Address – address of function on end of pipe
- Endpoint – endpoint for data
- Data – the payload of the packet

USB Connector/Signals

Pin out



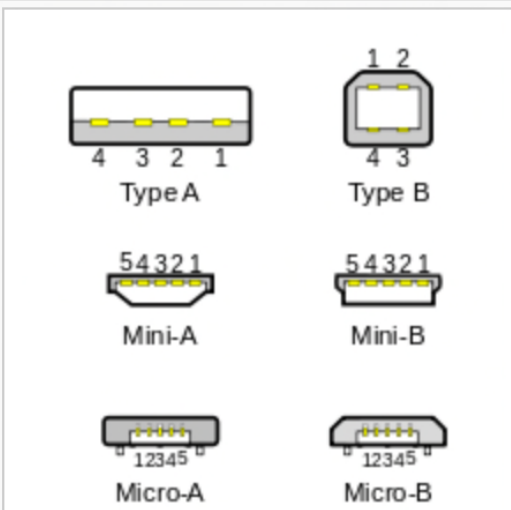
The standard USB A plug (left) and B plug (right) (male view)

Pin 1	V_{CC} (+5 V)
Pin 2	Data–
Pin 3	Data+
Pin 4	Ground

Electrical	
Signal	5 volt DC
Max. voltage	5.00±0.25 V
Max. current	500–900 mA @ 5 V (depending on version) 5 A for Battery Charging devices
Data	
Data signal	Packet data, defined by specifications
Width	1 bit
Bitrate	1.5/12/480/5,000 Mbit/s (depending on mode)
Max. devices	127
Protocol	Serial

http://en.wikipedia.org/wiki/Universal_Serial_Bus

USB Signals



Pinouts of Standard, Mini, and Micro USB plugs. The white areas in these drawings represent hollow spaces. As the plugs are shown here, the USB logo (with optional letter A or B) is on the top of the overmold in all cases.[*contradiction*][23]

USB 1.x/2.0 standard pinout

Pin	Name	Cable color	Description
1	VBUS	Red	+5 V
2	D–	White (gold*)	Data –
3	D+	Green	Data +
4	GND	Black (blue*)	Ground

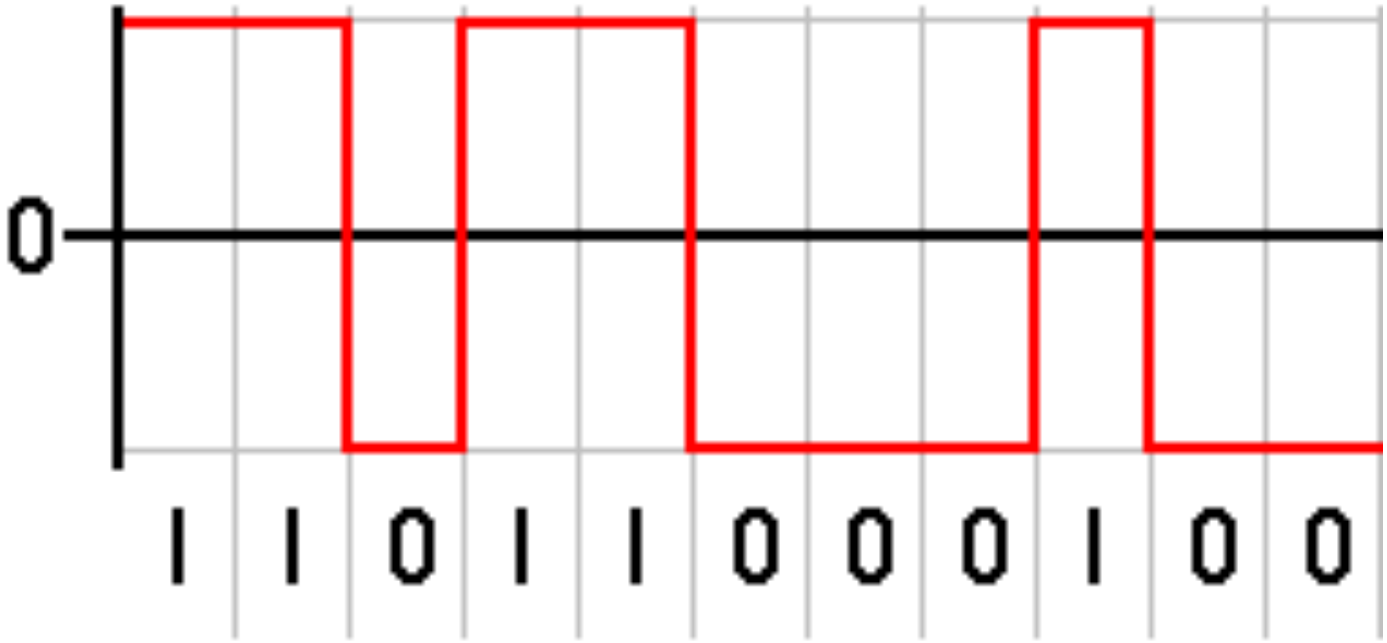
- *Some manufacturers use

USB 1.x/2.0 Mini/Micro pinout

Pin	Name	Cable color	Description
1	VBUS	Red	+5 V
2	D–	White	Data –
3	D+	Green	Data +
4	ID	None	Permits distinction of host connection from slave connection * host: connected to Signal ground * slave: not connected
5	GND	Black	Signal ground

http://en.wikipedia.org/wiki/Universal_Serial_Bus

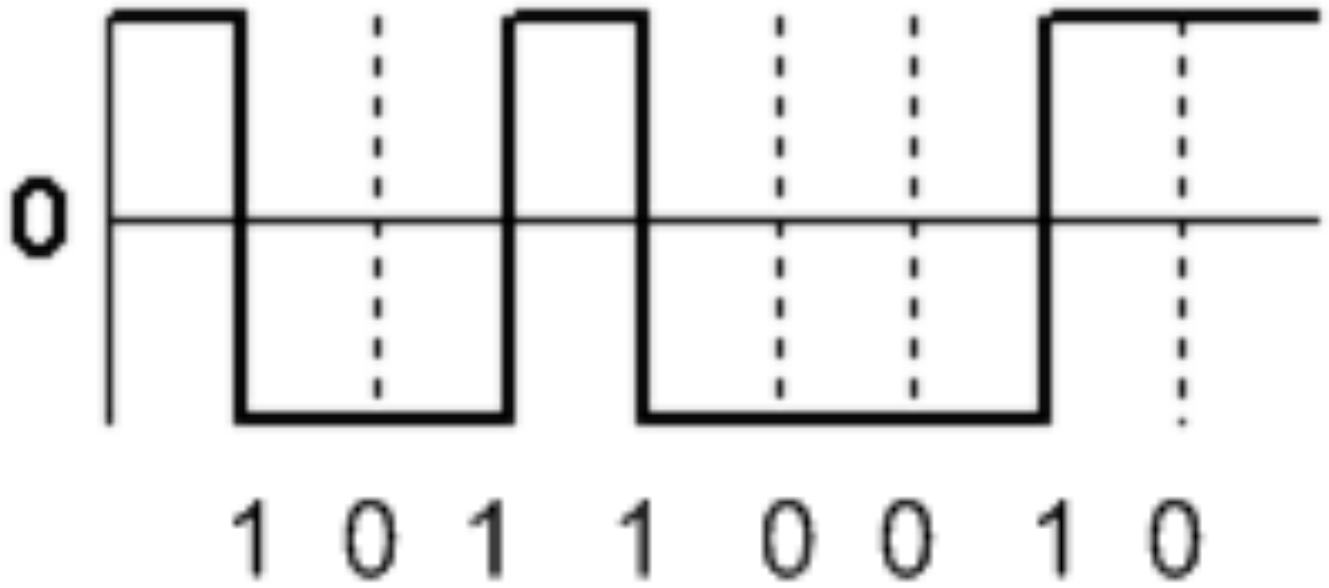
NRZ-Non-Return to Zero



Serial Data at Baseband

http://en.wikipedia.org/wiki/Universal_Serial_Bus

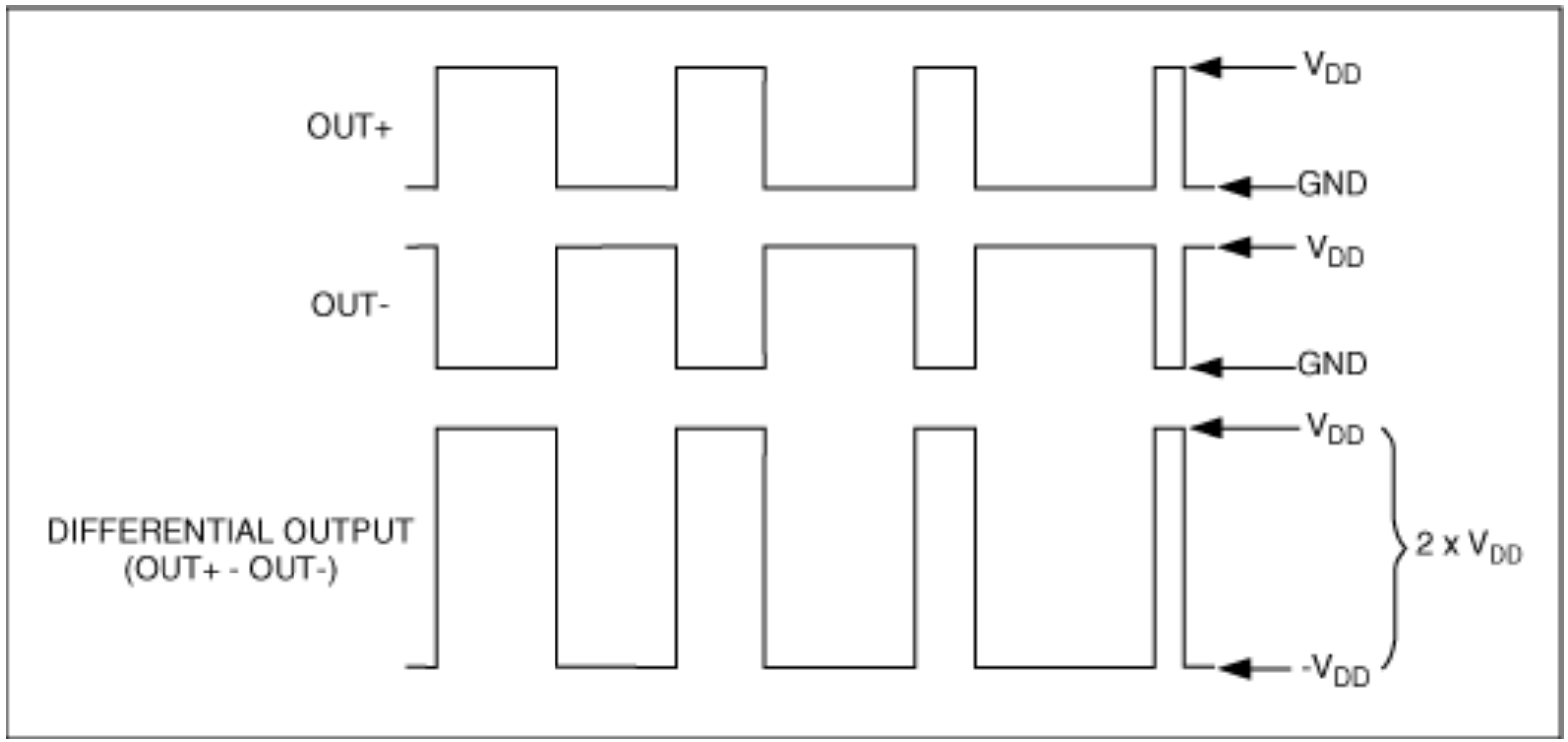
NRZI-Non-Return to Zero Inverted



Transitions are 1, Constant are 0

http://en.wikipedia.org/wiki/Universal_Serial_Bus

Differential Signal (Dual-rail)



Common-mode Noise Rejection

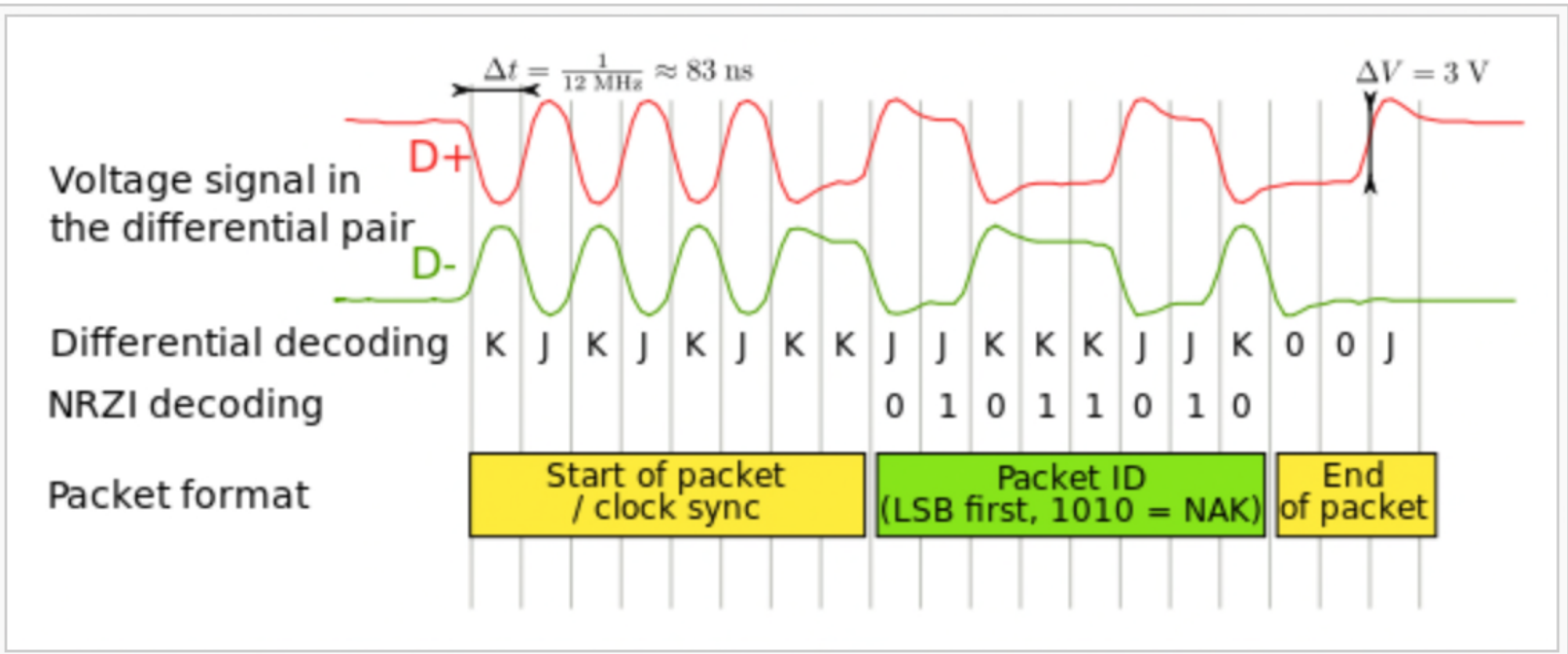
Better for High BW Transmissions

http://en.wikipedia.org/wiki/Universal_Serial_Bus

USB Signaling

- Based on NRZI Differential Encoding
- Asynchronous Transmission uses a SYNC Frame
- Receiver Detects SYNC Frame and Starts Local CLK
  - Clock Data Recovery (CDR) Extracts CLK from Data
  - Usually a PLL or DLL

USB Sync Frame



Example of a Negative Acknowledge packet transmitted by USB 1.1 Full-speed device when there is no more data to read. It consists of the following fields: clock synchronization byte, type of packet and end of packet. Data packets would have more information between the type of packet and end of packet.

http://en.wikipedia.org/wiki/Universal_Serial_Bus

USB Line States

Line State. These signals reflect the current state of the single ended receivers. They are combinatorial until a "usable" **CLK** is available then they are synchronized to **CLK**. They directly reflect the current state of the **DP** (LineState[0]) and **DM** (LineState[1]) signals:

DM	DP	Description
0	0	0: SE0
0	1	1: 'J' State
1	0	2: 'K' State
1	1	3: SE1

NRZI Encoding:Non-Return to Zero Inverted

J→K AND K→J Indicates a Zero

K→K AND J→J Indicates a One

p. 12 UTMI spec

CDR

INTRODUCTION AND BASICS

Why Clock and Data Recovery Circuits?

In many systems, data is transmitted or retrieved without any additional timing reference. For example, in optical communications, a stream of data flows over a single fiber with no accompanying clock, but the receiver is required to process this data synchronously. Therefore, the clock or timing information must be recovered from the data at the receiver.

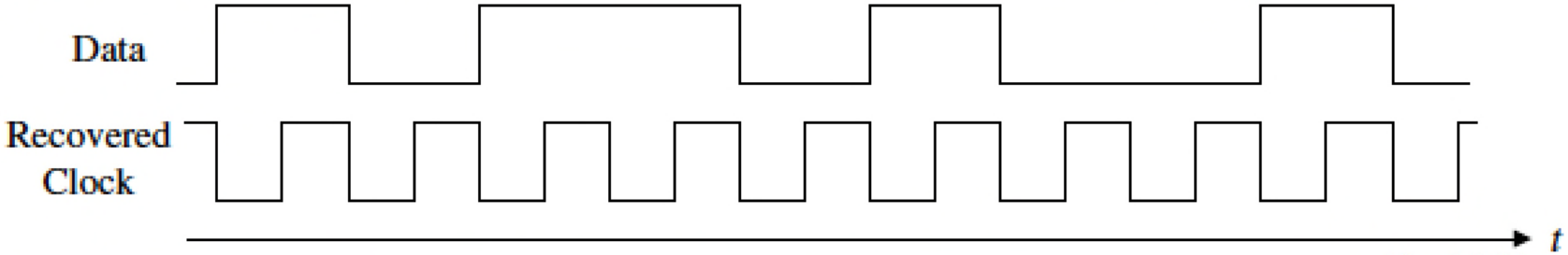


Fig. 4.2-01

Most all clock recovery circuits employ some form of a PLL.

http://users.ece.gatech.edu/jskenney/L200-CDR-I(2UP).pdf

CDR Architecture

CLOCK RECOVERY ARCHITECTURES AND ISSUES

Clock Recovery Architectures

From the previous considerations, we see that clock recovery consists of two basic functions:

1.) Edge detection

2.) Generation of a periodic output that settles to the input data rate but has negligible drift when some data transitions are absent.

Conceptual illustration of these functions:

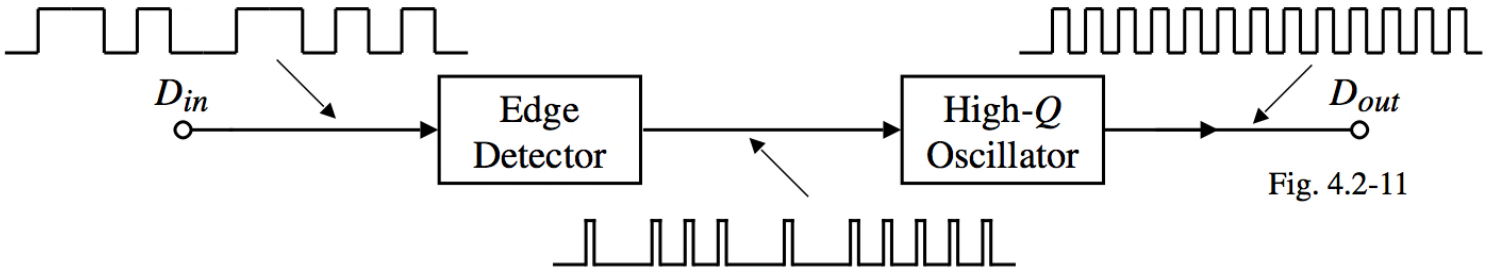


Fig. 4.2-11

In essence, the high-Q oscillator is "synchronized" with the input transitions and oscillates freely in their absence. Synchronization is achieved by means of phase locking.

http://users.ece.gatech.edu/jskenney/L200-CDR-I(2UP).pdf

CDR Edge Detection

Edge Detection

CRC circuits require the ability to detect both the positive and negative transitions of the incoming data as illustrated below,

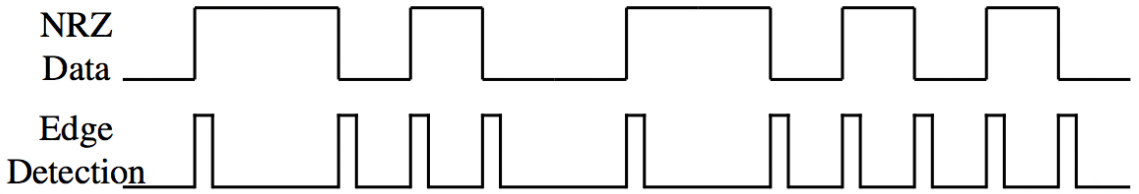


Fig. 4.2-05

Methods of edge detection:

1.) EXOR gate with a delay on one input.

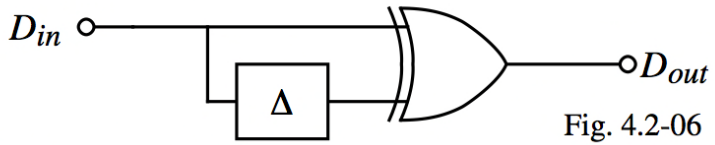


Fig. 4.2-06

2.) A differentiator followed by a full-wave rectifier.

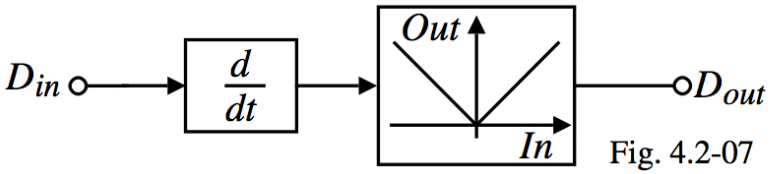


Fig. 4.2-07

http://users.ece.gatech.edu/jskenney/L200-CDR-I(2UP).pdf

CDR PLL-Based

Phase Locked Clock Recovery Circuit

Circuit:

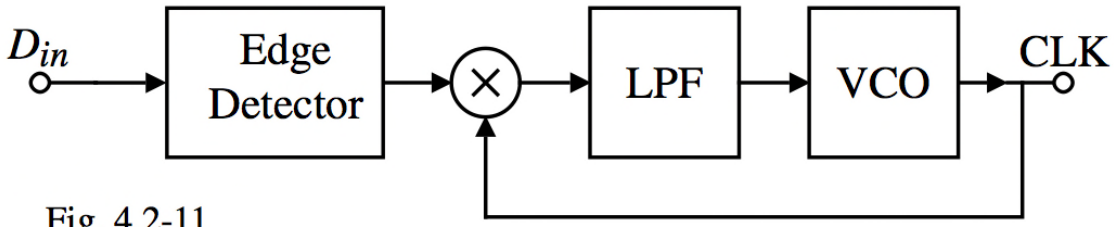


Fig. 4.2-11

Operation:

1.) Assume the input data is periodic with a frequency of $1/T_b$ (Hz).

2.) The edge detector doubles the frequency causing the PLL to lock to $2/T_b$ (Hz).

3.) If a number of transitions are absent, the output of the multiplier is zero and the control voltage applied to the VCO begins to decay causing the oscillator to drift from $1/T_b$ (Hz).

4.) To minimize the drift due to the lack of transitions,

τ_{LPF} >> Maximum allowable interval between consecutive transitions.

5.) The result is a small loop bandwidth and a narrow capture range. Fortunately, most communication systems guarantee an upper bound of the allowable interval between consecutive transitions by encoding the data.

http://users.ece.gatech.edu/jskenney/L200-CDR-I(2UP).pdf

USB Host-Attachment Clock Data Recovery

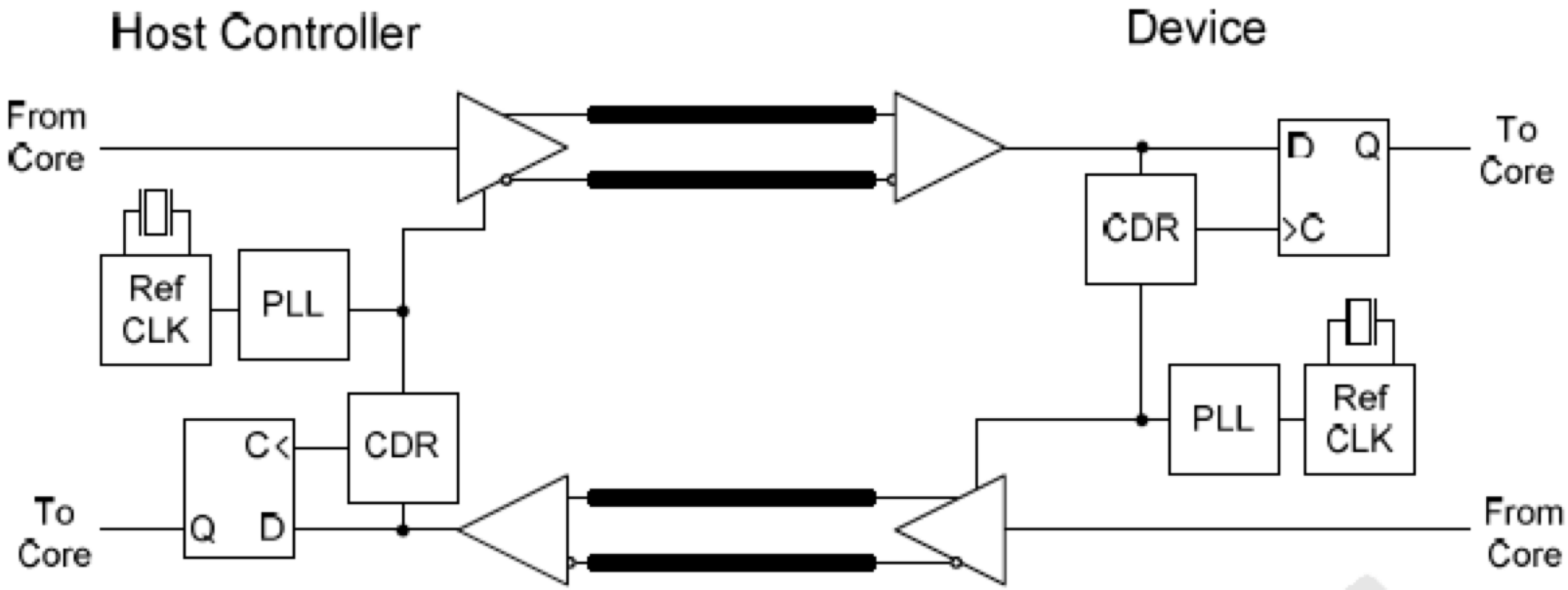


Figure 1: SuperSpeed USB clocking and jitter architecture

Source: USB 3.0 CDR Model White Paper, Revision 0.5, HP, Intel, Microsoft, NEC, ST-NXP, TI, Jan. 15, 2009

USB CDR

In USB the receiver recovers the clock from the data by digitally adjusting the phase of the local clock to try to match the phase of the incoming data as closely as possible. The

difference of the phase of the recovered clock and the data is a timing error, or jitter. The clock recovery circuit is shown in the following block diagram.

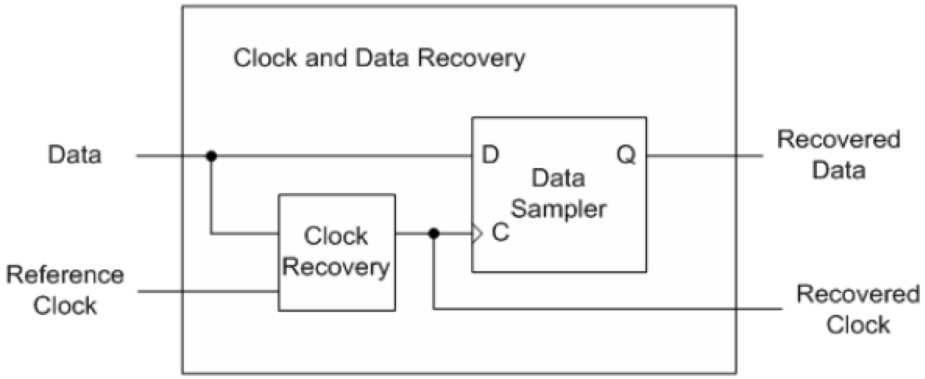


Figure 3: Receiver clock and data recovery

Source: USB 3.0 CDR Model White Paper, Revision 0.5, HP, Intel, Microsoft, NEC, ST-NXP, TI, Jan. 15, 2009

USB 2.0 Clocks

5.12.2 USB Clock Model

Time is present in the USB system via clocks. In fact, there are multiple clocks in a USB system that must be understood:

- Sample Clock: This clock determines the natural data rate of samples moving between client software on the host and the function. This clock does not need to be different between non-USB and USB implementations.
- Bus Clock: This clock runs at a 1.000 ms period (1 kHz frequency) on full-speed segments and 125.000 µs (8 kHz frequency) on high-speed segments of the bus and is indicated by the rate of SOF packets on the bus. This clock is somewhat equivalent to the 8 MHz clock in the non-USB example. In the USB case, the bus clock is often a lower-frequency clock than the sample clock, whereas the bus clock is almost always a higher-frequency clock than the sample clock in a non-USB case.
- Service Clock: This clock is determined by the rate at which client software runs to service IRPs that may have accumulated between executions. This clock also can be the same in the USB and non-USB cases.

In most existing operating systems, it is not possible to support a broad range of isochronous communication flows if each device driver must be interrupted for each sample for fast sample rates. Therefore, multiple samples, if not multiple packets, will be processed by client software and then given to the Host Controller to sequence over the bus according to the prenegotiated bus access requirements. Figure 5-17 presents an example for a reasonable USB clock environment equivalent to the non-USB example in Figure 5-16.

Source: USB 2.0 Standard

USB 2.0 Clocks

5.12.3 Clock Synchronization

In order for isochronous data to be manipulated reliably, the three clocks identified above must be synchronized in some fashion. If the clocks are not synchronized, several clock-to-clock attributes can be present that can be undesirable:

- Clock Drift: Two clocks that are nominally running at the same rate can, in fact, have implementation differences that result in one clock running faster or slower than the other over long periods of time. If uncorrected, this variation of one clock compared to the other can lead to having too much or too little data when data is expected to always be present at the time required.
- Clock Jitter: A clock may vary its frequency over time due to changes in temperature, etc. This may also alter when data is actually delivered compared to when it is expected to be delivered.
- Clock-to-clock Phase Differences: If two clocks are not phase locked, different amounts of data may be available at different points in time as the beat frequency of the clocks cycle out over time. This can lead to quantization/sampling related artifacts.

The bus clock provides a central clock with which USB hardware devices and software can synchronize to one degree or another. However, the software will, in general, not be able to phase- or frequency-lock precisely to the bus clock given the current support for "real time-like" operating system scheduling support in most PC operating systems. Software running in the host can, however, know that data moved over the USB is packetized. For isochronous transfer types, a unit of data is moved exactly once per (micro)frame and the (micro)frame clock is reasonably precise. Providing the software with this information allows it to adjust the amount of data it processes to the actual (micro)frame time that has passed.

Note: For high-speed high-bandwidth endpoints, the data exchanged in the two or three transactions per microframe is still considered to belong to the same "single packet." The large amount of data per packet is split into two or three transactions only for bus efficiency reasons.

Source: USB 2.0 Standard

USB 2.0 Host Interface

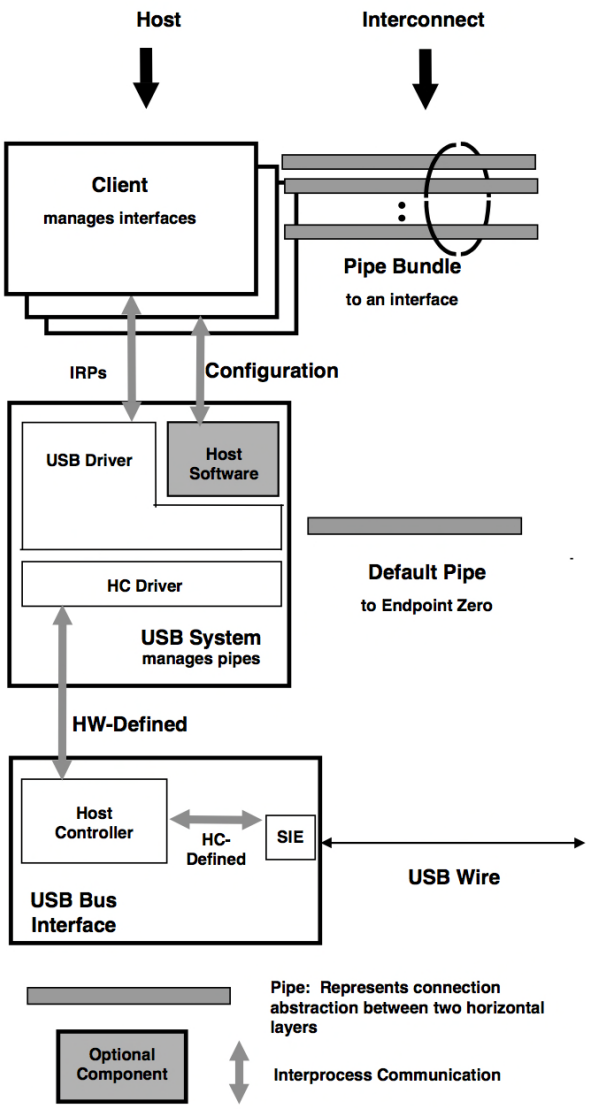


Figure 10-2. Host Communications

Source: USB 2.0 Standard

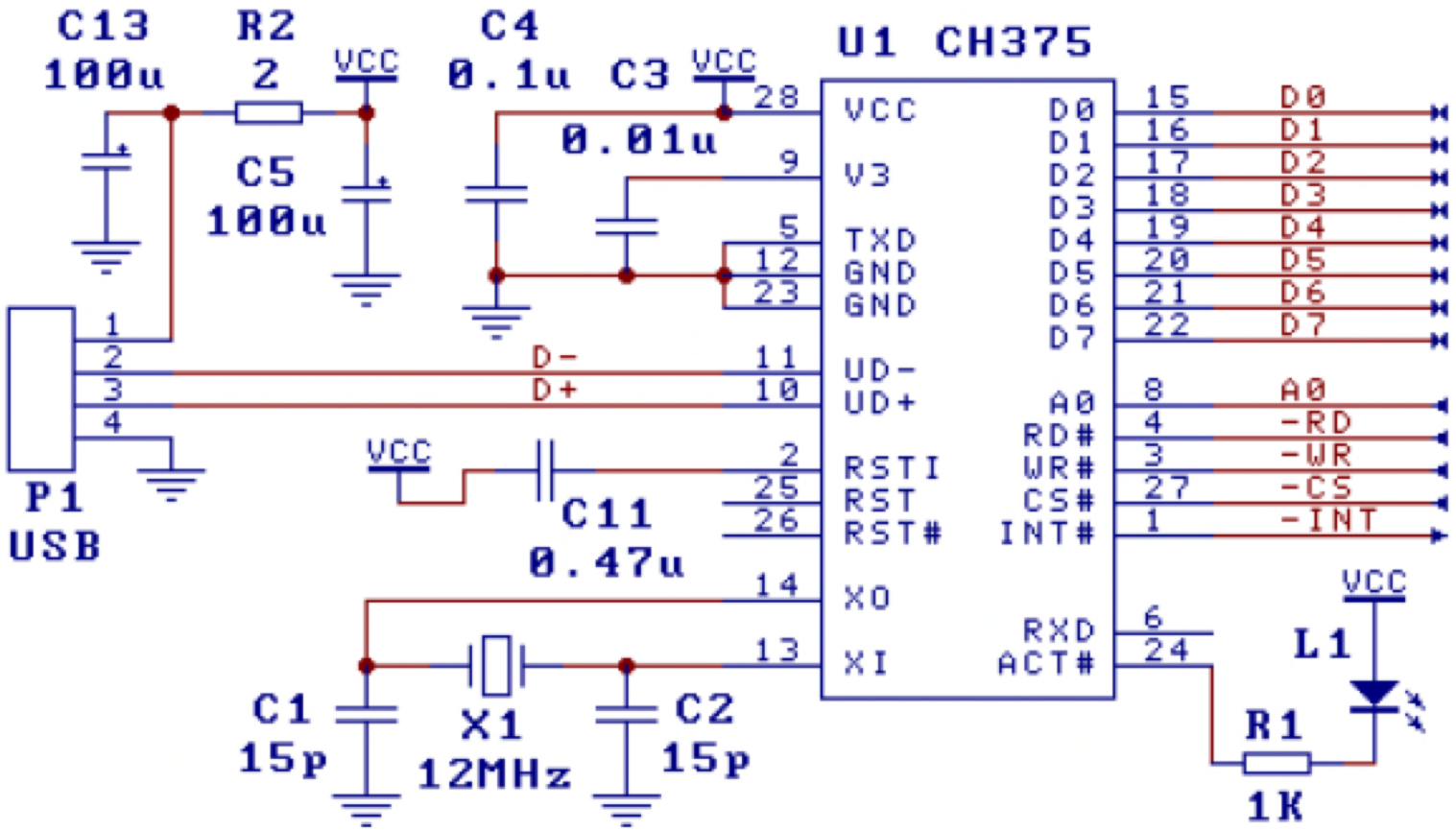
USB 2.0 SIE (SERDES)

10.2.2 Serializer/Deserializer

The actual transmission of data across the physical USB takes places as a serial bit stream. A Serial Interface Engine (SIE), whether implemented as part of the host or a USB device, handles the serialization and deserialization of USB transmissions. On the host, this SIE is part of the Host Controller.

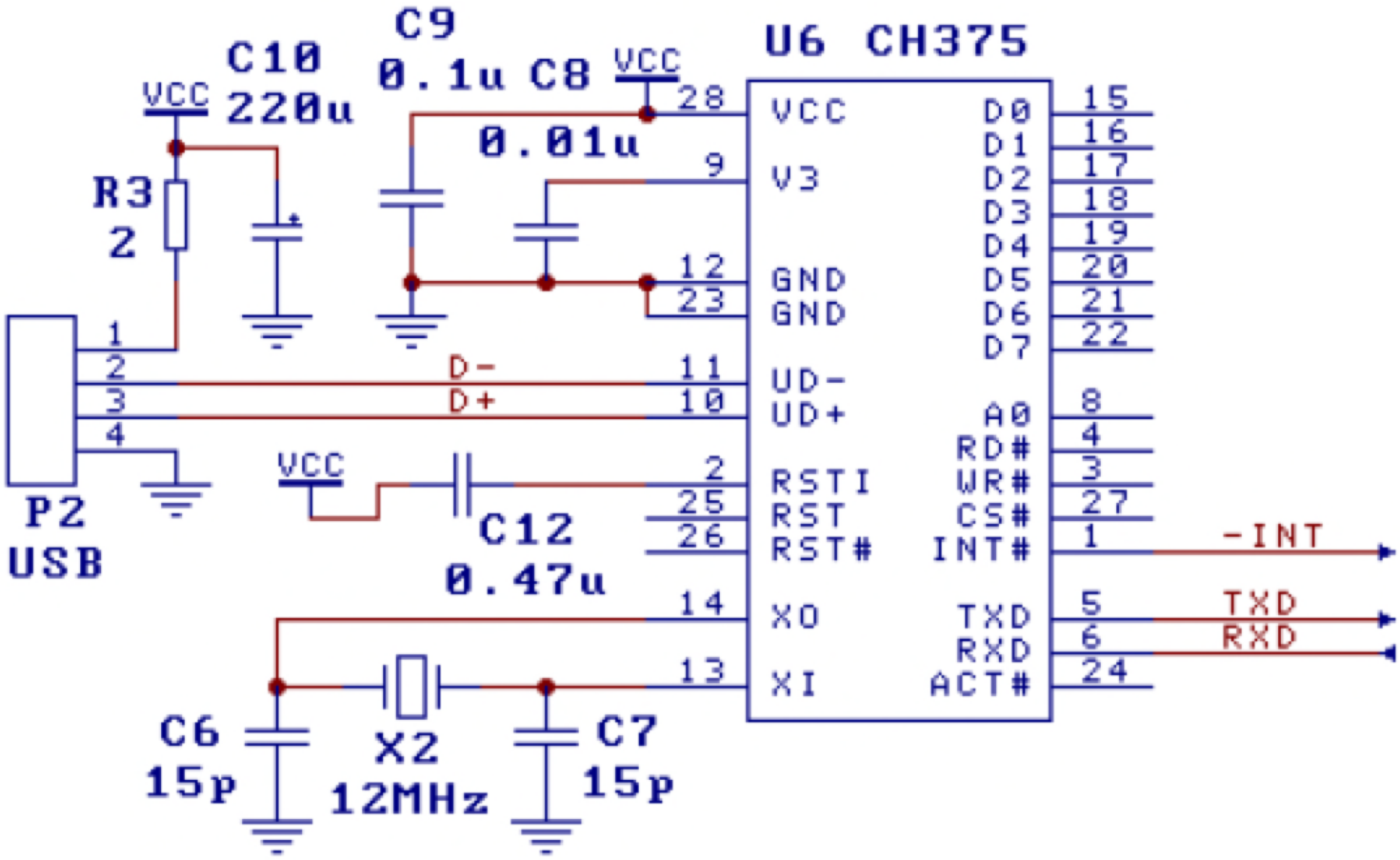
Source: USB 2.0 Standard

USB Example Interface



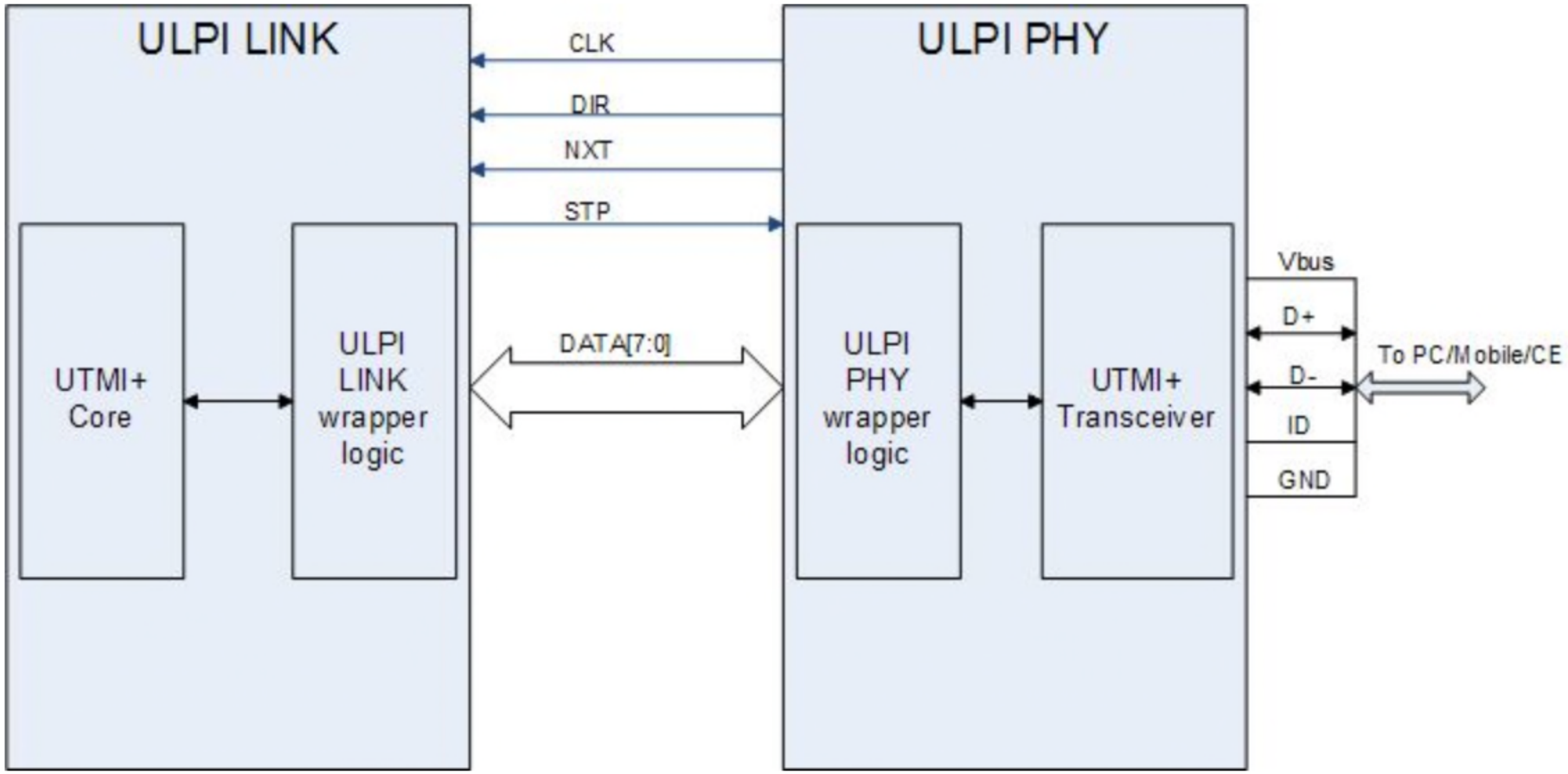
Source: http://docs.teguna.ro/CH375DS1.pdf, Romainian IC Vendor

USB Example Interface



Source: http://docs.teguna.ro/CH375DS1.pdf, Romainian IC Vendor

ULPI Figure



Source: http://www.ulpi.org

PHY – Physical Layer portion, conversion of serial USB signals to logic signals appropriate for processing by a CPU

UTMI

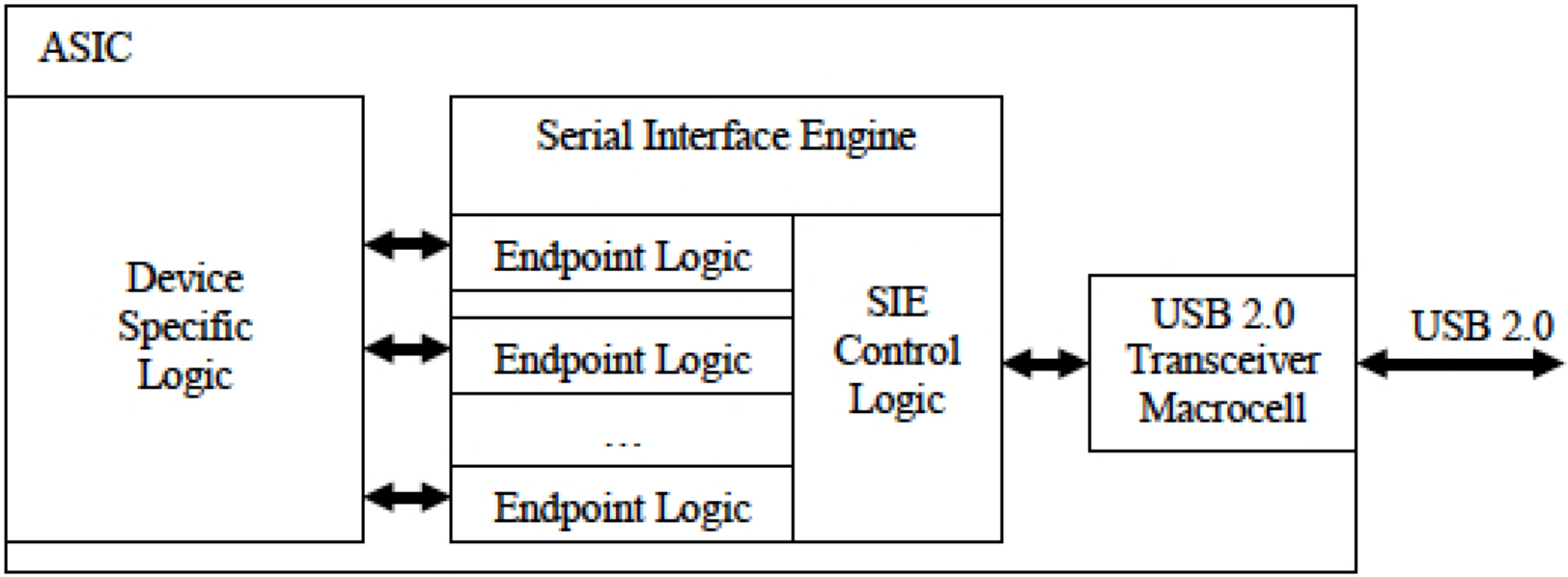


Figure 1: ASIC Functional Blocks

Source: p. 9, Fig. 1, USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, v. 1.05, 29 Mar 2001, Intel Corp.
http://www.intel.com/technology/usb/download/2_0_xcvr_macrocell_1_05.pdf

UTMI

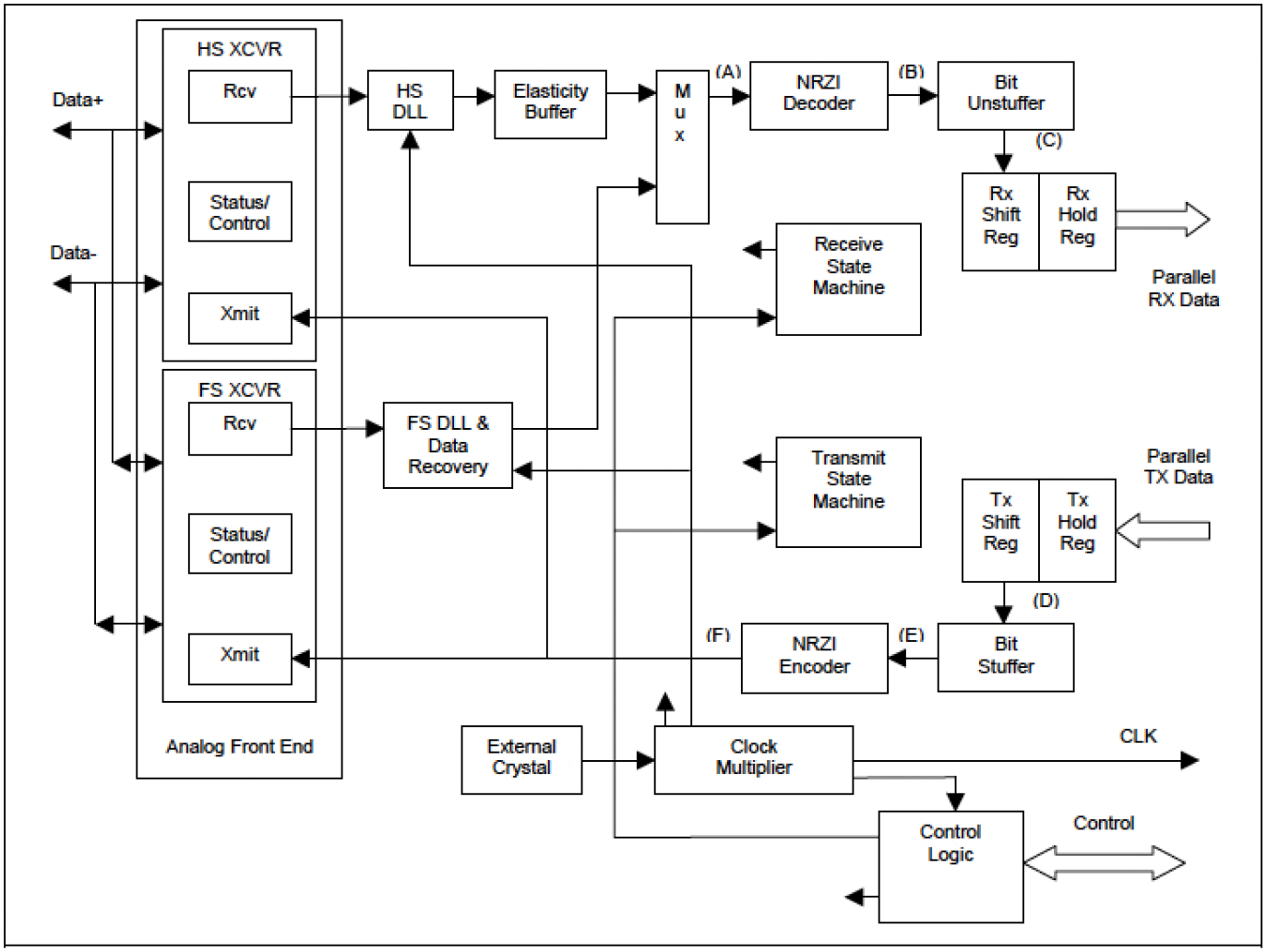


Figure 2: UTM Functional Block Diagram

Source: p. 11, Fig. 1, USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, v. 1.05, 29 Mar 2001, Intel Corp.
http://www.intel.com/technology/usb/download/2_0_xcvr_macrocell_1_05.pdf

UTMI Blocks

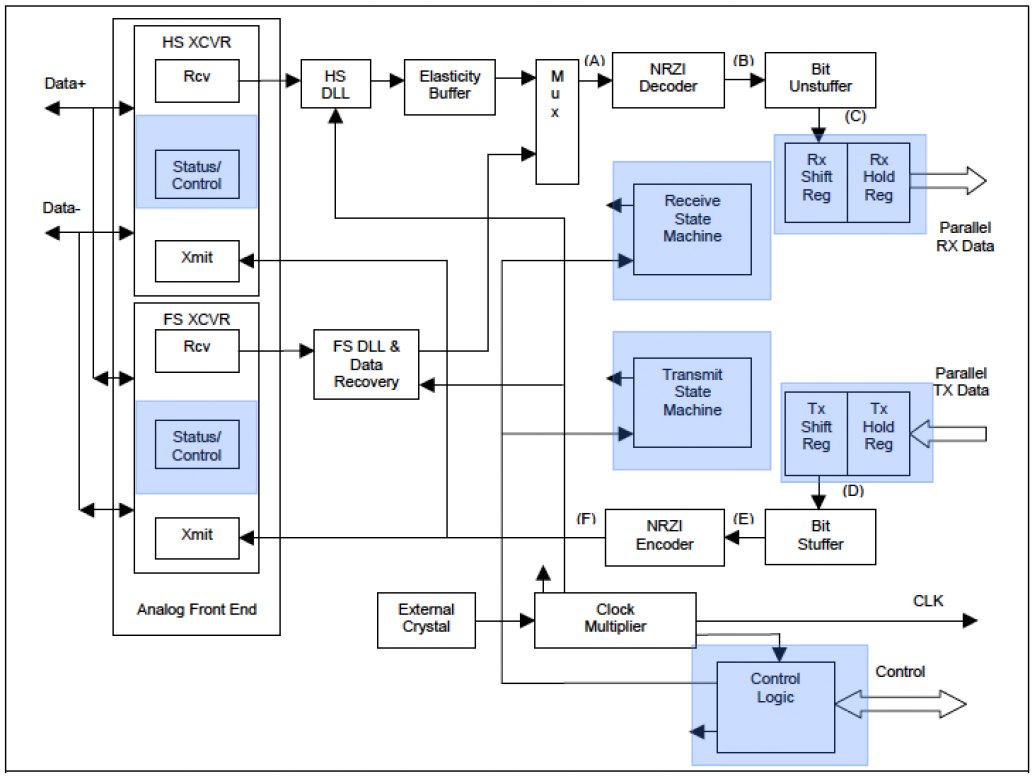


Figure 2: UTM Functional Block Diagram

These logic blocks require CLK signals to operate
can use same or a derivative of same SYS