Hardware

The hardware interface for the USB is quite a challenge for someone unfamiliar with fast digital electronics. IFR opted to use an ISA development card from RS to build the interface on. A placement student completed the initial build. - The circuit design was based very closely on that of the real control board that we need to emulate. Philips provide a reasonable level of documentation in support of the PDIUSBD12 but some rework was necessary due to the rather ambitious description in some areas of the D12 data sheet

Philips also provides some example firmware that will compile and run on any MS DOS machine. The firmware is designed to be used with the D12 evaluation kit. As a result some modification of the C source was necessary in order to have any hope of being able to communicate with the D12 and the USB host beyond.

The firmware provided was written in a very terse structured style with lots of magic numbers instead of self-documenting variable names. Converting and understanding this dos application was initially quite a challenge. The problem was compounded by a complete lack of useful tools at the host end of the USB link all tools available to the author at the time relied on the device being capable of enumerating correctly.

USBview, another Microsoft utility provides quick test to check whether a device has enumerated correctly or not.

Debugging the Enumeration

Debugging the enumeration sequence can be very difficult to achieve, because of the time constraints placed by the USB Spec on the device. - Using simple printf statements within command servicing routines introduced delays sufficient to break the enumeration.

The other main technique is to use some form of debug utility at the host end of the bus. - This can be achieved by installing debug builds of the key windows drivers such as USB.sys and hid.sys. Using wdeb.exe installed as a VXD allows the debug prints generated by the drivers to be sent via Ethernet or serial to an external terminal program (Rterm98.exe) in real time. This configuration allows the user a good view of relevant system events.

Once the descriptors were verified as being correct on the DOS firmware, a reason had to be found for enumeration failures being observed. In most cases connecting the Device caused the host to "lock up" and eventually bug check. This is very unfortunate behavior, and made saving a debug print out the enumeration very difficult until the remote terminal was added to the configuration.

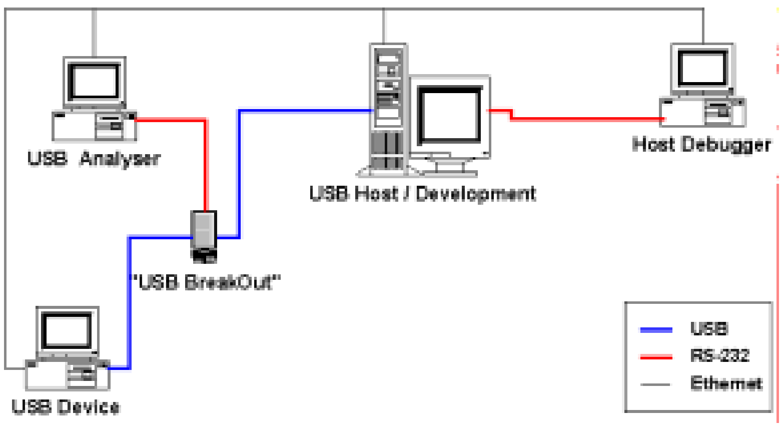

Figure 1 The development configuration required to aid debugging

To provide an example of the desired behavior, a generic USB mouse was purchased in an effort to determine why the enumeration failed. That the mouse enumerated correctly was a good indication that the device firmware was causing the failures, rather than software at the host end.

The next approach taken was to attempt to capture the actual USB bus traffic and decode it.

DIAGRAM2 – USB Analyser Diagram from note book

The main problem associated with this approach was the capture speed. Because of the limited size of the capture memory it is crucial to trigger the analyser at precisely the correct moment. This trigger was initially achieved using the programmable clock output of the D12, though it was realised that the trigger would have to be fired at the beginning of a non - idle packet. This required some detection logic to be added to the sampling hardware. As a result of this new development IFR decided to hire a USB analyser.

Figure 2 CATC USB Analysers

This piece of equipment proved invaluable throughout the period it was available. It was found that the extra bytes being sent from the device in error were causing the host to bug check. This led to the realisation that the problem all along had been that the compiler for the device firmware was packing the structures used to store the device descriptors as word aligned, rather than byte aligning them. This was soon rectified, and the device enumerated correctly.

A small test application was built on top of the "Bulkusb.sys" driver from Microsoft's examples. This provided an efficient means of testing the prototype system.

DIAGRAM4 – SCREENSHOTS OF BULKUSB AND DOS FIRMWARE?

Now that the hardware is working and I have proved the feasibility of data transfer using the USB I need to set about building a GPIB capable system.

Firmware

The steps to convert the known working Dos system to the VxWorks version were not huge leaps. Instead the structured code was rewritten using an OO style (i.e. Using Objects rather than object orientation), this allowed a swift conversion from procedures to classes and methods when the code was ported to VxWorks This technique has been reasonably successful, though it could be said that the resulting object is more complex than a pure OO design would have produced.

Having created the objects in VxWorks, it was then necessary to wrap them in a USB class that could be integrated with the rest of the instrument firmware.

DIAGRAM5 – From DOS to VxWorks?

To this end, a snapshot of the Lazarus firmware was taken, and then recompiled to run on the 486 PC platform as opposed to the control boards now available to senior project developers.

This ensured that the new USB classes would not interfere with the main project development, and isolated the core code base from problems induced by adding a new hardware interface.

Because of the way that the software had been designed to support the GPIB interface it was reasonably easy to add the USB class into the system. The USB class was altered to closely resemble the GPIB class in the way that Data streams were handled and "connected" to the main instrument core.

As can be seen from the diagram below, the resulting system is a clean design that allows the USB interface to be developed with no requirement for a deep understanding of the instrument functions. This should aid reuse in future projects.

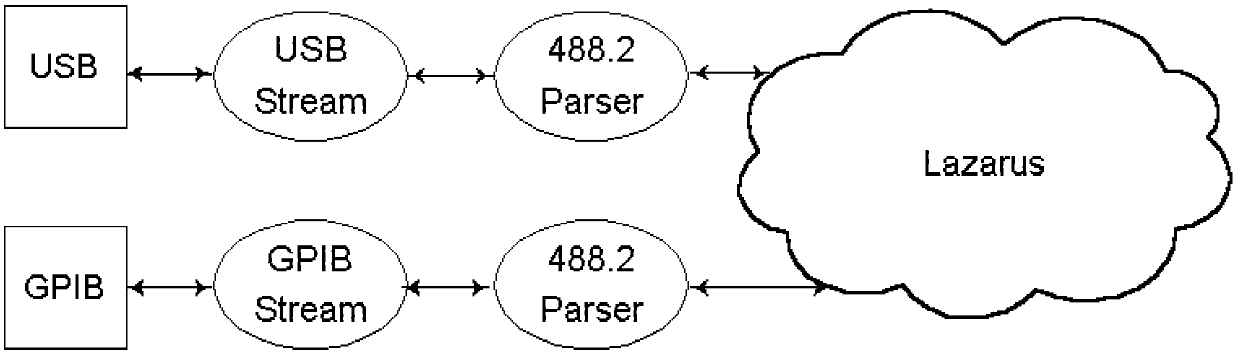

Figure 3 Integrating USB support into the Instrument Firmware

The firmware has now been compiled, and runs with few, if any, problems on the 486 PC development platform. At a later point in the project the code developed in Bristol will be merged with that of the Stevenage site and then a full test of the firmware can be completed with the control board. This will ensure minimum integration problems in the final stages of the project.

. _To-do_
DIAGRAM2 – USB Analyser Diagram [from notebook] WDM compiler chain
DIAGRAM4 – SCREENSHOTS OF BULKUSB AND DOS FIRMWARE?
DIAGRAM5 – From DOS to VxWorks

App - BulkUSB.sys
Zeus talker

Word Count: 1067