DD X

STOCKHOLM UNIVERSITY

DEPARTMENT OF PHYSICS

A GENERIC PROGRAMMABLE GPIB-INTERFACE

S. AGNVALL, R. HELLMANS, P. STENSTRÖM

USIP Report 95 - 04

May 1995

Stockholms universitet/Reproenheten 1996

A generic programmable GPIB-interface

S. Agnvall, R. Hellmans, P. Stenström

Department of Physics, University of Stockholm

Box 6730, S-113 85 STOCKHOLM, Sweden

Abstract

A generic interface to the IEEE-488.1 bus is described. The interface is designed to simplify the connection of an instrument to this bus. It is also shown that the generic programmable interface is shortening the time for the development phase of an application.

Introduction

When designing computerised data acquisition and/or measuring systems one is often faced with the problem of how to bring information from one device to another. Control commands and data could be exchanged via point-to-point links or via a common system bus. In both cases the devices will need interfacing to be able to interconnect. Whenever feasible it is advisable to stay with standardised commercial products, but sometimes this is not possible. In those cases it is often necessary to design special purpose equipment, which is a time consuming task. However, since most solutions contain common components we have designed a general purpose instrument interface incorporating such components, i.e. an interface capable of being adapted to a large class of different situations. This general approach is more viable if the interfaces are built from programmable components, such as programmable logic, memories and microcontrollers. The programmability can then adapt the generic interface to most of the requirements, minimizing the amount of application specific hardware.

The IEEE-488.1-bus, commonly known as the General Purpose Interface Bus, GPIB, is in wide use in industry today as a way to connect different instruments to form local systems for data acquisition or control, provided the data transfer requirement does not exceed 1MB/s. We will here present a GPIB based solution for a programmable generic interface to be used when developing new instruments or to aid in the design of prototypes for data acquisition/control systems. The presence of a micro controller will allow the interface to perform modest computations. This is often an advantage, since modern data acquisition systems relies more and more on computations in order to reduce the amount of data. Including programmable logic will also allow the implementation of fast but simple computations as well as a direct memory access function.

The MUltifunctional GPIB-interface, MUG system or MUG-board, is an interface between the GPIB-bus and some other system, like a data acquisition or control system. This second system will from now on be referred to as an 'application'. MUG is not built for a specific purpose other than just being a link between an application and the GPIB-bus. However it is a very versatile link, implemented as a generic interface with computational power. The philosophy in this design is to make this device as universal as possible and yet simple to use. There often exists a trade-off between universality and simplicity. We have tried to achieve as much as possible of both.

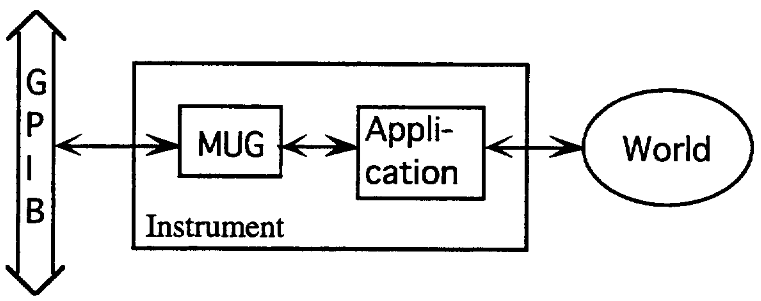


Fig 1: MUG and its environment

The microcontroller makes it possible to implement complex algorithms within the interface. This facility may be used for pre-processing data in a data acquisition system. Alternatively, in a control system one might implement controlling algorithms in the MUG-system and send the status of the system over GPIB. Some of the features of the MUG system include: A/D-D/A conversion, Direct Memory Access-mode (DMA-mode) and local buffering of data. The last two features are important if one wishes to transfer data from an application at higher speeds than the local microcontroller can handle. From now on the term, 'instrument', will be used to denote a superset consisting of an application and MUG, cf. fig. 1 above.

Description of the system

MUG consists of three functional blocks: a GPIB-interface, logic and a microcontroller. The GPIB interface is responsible for connecting the MUG-system electrically and functionally to GPIB, therefore consisting of a GPIB-controller and transceivers. To achieve proper functionality of the GPIB interface it is necessary to program the GPIB-controller. This is done by writing control words to registers located inside the controller. Some of them hold the primary and secondary talk/listen addresses while others are used to transfer data to and from the MUG-system.

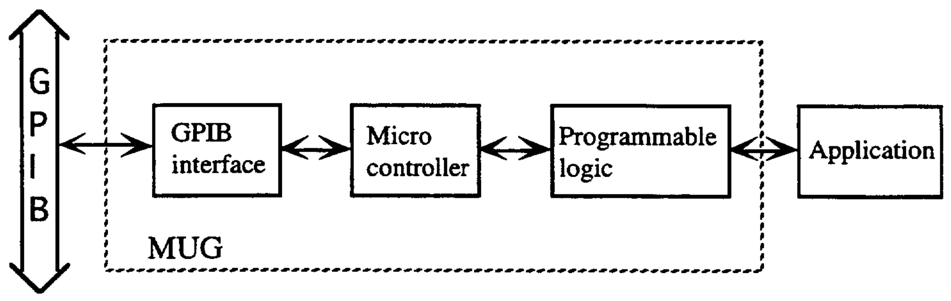


Fig 2: Functional overview of the MUG system

The programmable logic in MUG is divided in two parts. The system part, which is internal to MUG, is dedicated to initialising the GPIB transmissions and to decode the internal address space. This logic, which can not be seen from outside MUG, is fixed. By fixed we mean that certain logic must be contained within this circuit in order for MUG to work properly. The application logic is part of the interface to the application. It can be used to generate flexible interfacing to the application and act as a DMA controller. The application type logic needs to be programmed every time power is switched on. It may seem as a major drawback but in fact it is not. Connected to the application logic is a small memory which holds the configuration. When power is switched on this memory loads its contents into the application logic. There is also a possibility to reprogram the application logic 'on the fly' via GPIB and the microcontroller. This possibility adds to the reconfigurability of the MUG-system.

The microcontroller is responsible for initialising, managing and supervising the MUG-system. It is programmed to parse and execute commands sent to MUG over GPIB. This includes commands used to program the logic in MUG and to run self-tests. The microcontroller could also, as has already been mentioned, be used for computational tasks, such as pre-processing of incoming data. Of course, there are cases where the microcontroller is not powerful enough, i.e. when computing becomes too demanding. Should this situation occur, computation would be transferred outside MUG to the application, and the microcontroller would be used for simpler tasks as managing and supervising. Such tasks are well suited for the microcontroller. Only when computing requirements are modest it is advisable to use the microcontroller.

Hardware

Architecture

The vital parts of MUG are the 8-bit microcontroller and the Field Programmable Gate Array (FPGA). We have chosen to work with Hitachi's H8/338 as microcontroller and Xilinx's Logic Cell Array (LCA) as FPGA. For the latter we use LCAs from Xilinx's 4000 series, from XC4002 to XC4010. Larger LCAs are not available in the PLCC84 package and can therefore not be used at the moment. The H8/338 microcontroller could also be replaced by the other two microcontrollers in the same series, namely H8/337 and H8/336. These are identical in the way they operate in MUG, but they differ from the H8/338 in the sense that they have smaller internal RAM, 1 kbyte for H8/337, or H8/336, instead of 2 kbyte as in the case for the H8/338. This difference is only relevant when internal RAM is to be used.

Around these components there are support circuits dealing with the communication with the outside world, an RS232-interface and a GPIB-controller, NAT7210, from National Instruments. This controller implements all the functions required by the IEEE-standard 488.1-1987 and meets the additional requirements and recommendations of the extension of this standard, IEEE-standard 488.2-1987. As was pointed out above, the GPIB controller's functions are controlled by internal registers. These are selected via three address pins, allowing for eight addresses. However, the GPIB controller has more than eight registers. Several addresses have extra registers which can only be accessed immediately after issuing a page-in command to the GPIB controller. The page-in command, and others, are issued by the microcontroller by writing to a special register in the GPIB controller. The main parts responsible for the functionality of the MUG system, and their connectivity is shown in fig 3 below.

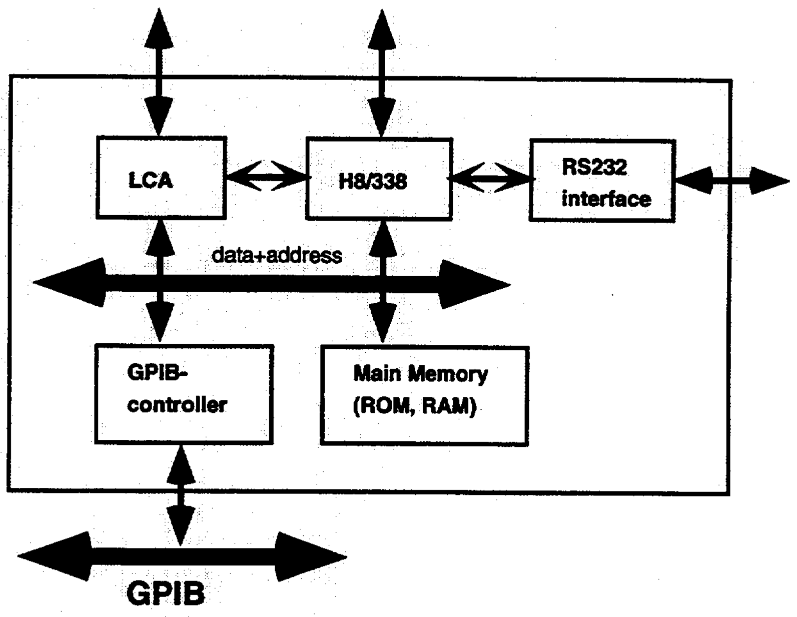


Fig. 3: The functional connectivity of the parts in the MUG system

Other support circuits are required for the operation of the system itself. This includes a Programmable Logic Device, PLD, used for decoding the address bus. This PLD should retain its configuration when power is switched off. Therefore we choose to use a non volatile PLD from Lattice, ispLSI 1016. This device is not as powerful as the FPGA but as flexible. Since both the LCA and the ispLSI are programmable in-circuit we are able to reconfigure the system dynamically. However, a part of the ispLSI is used by the MUG system for its' function. It is even possible to program the LCA, but not the ispLSI, when the system is running. The design of MUG allows these to be programmed via connectors on the MUG board, one for each device. An option to use a serial ROM to initially program the LCA has been added. This requires the capability to switch the programming mode for the LCA. A dip-switch is used for this purpose.

As an option a Static RAM (SRAM), which can not be reached directly by the microcontroller, can be connected to the LCA. This SRAM could be used as a buffer between an application and GPIB, should that be required. However, a small buffer is easily designed inside the LCA with the benefit of increasing the number of available pins on the interface to the application since in this case they are not needed for interfacing the external memory. Connected to the microcontroller are 32 kbyte of ROM and 32 kbyte of SRAM.

On the internal data- and addressbus we have the microcontroller, the LCA, ROM, RAM and the GPIB-controller, as seen in the picture above. In order to add DMA-capability for the data transfer between the application and GPIB we have to divide the data bus in two independent domains where the first domain contains the microcontroller and the memories, RAM and ROM, and the second domain contains the LCA and the GPIB-controller.

Application Interface

The interface between an application and MUG consists of a 31 bit general I/O-port from the LCA, a parallel I/O-port of 8 bits and a parallel input-only port of 8 bits coming from the microcontroller. Two RS232-ports constitute the serial interface. The input-only port can be used as an input port for 8 ADCs located inside the microcontroller. Two of these bits can also be defined as analogue output for DACs internal to the microcontroller.

Direct Memory Access

Direct Memory Access is, as was mentioned above, achieved in the MUG system by using a transceiver circuit to decouple the data bus between the microcontroller and the GPIB-controller. In addition the LCA must be programmed to act as a DMA-controller. A functional sketch of how this is done can be seen in fig. 4 below, which implements write operations only. Inside the LCA a state machine has been designed to control the DMA process. Care must be taken not to deadlock the GPIB-system in this mode without a possibility to interrupt the DMA transfer. For this reason the microcontroller must be able to take control over the system, on request. The state machine is designed to send a byte to GPIB via DMA and then to check if the microcontroller wants control of GPIB. If not, the cycle is repeated all over again. This means that the Controller In Charge (CIC) on GPIB can still send interrupts to the MUG system, which are serviced by the microcontroller, allowing the interface to respond to events such as unlisten, IFC et c.

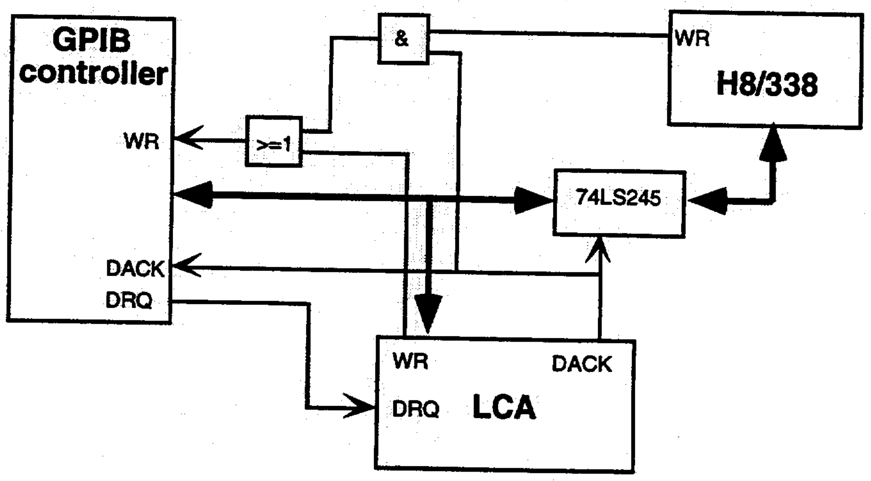


Fig. 4: Realisation of Direct Memory Access. Only write signal is shown.

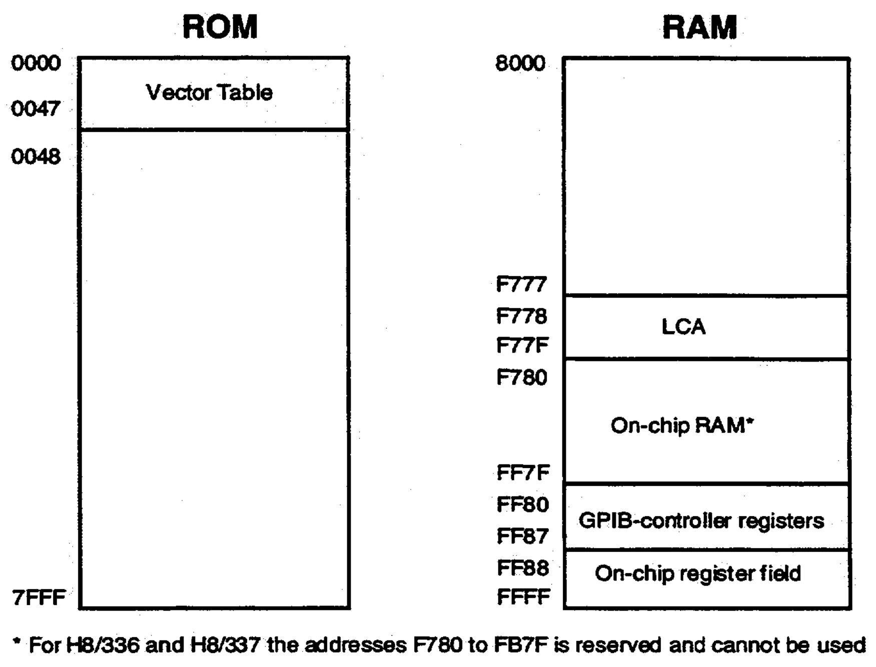
There are two ways for the MUG system to enter DMA mode. First, DMA could be requested by another GPIB device causing MUG to change mode. Second, it can be initiated inside MUG by the microcontroller itself. When DMA is accepted by another GPIB device the LCA takes control over the DMA-transfer.

DMA-transfer is initiated, or requested, inside MUG by the GPIB-controller by asserting the DRQ signal. The LCA acting as a DMA-controller thus asserts the DACK signal and thereby signalling to the transceiver circuit, 74LS245, to release the data bus between itself and the GPIB-controller by putting its outputs in tristate (on the left side of the 74LS245 in the figure above). Asserting the DACK signal also shuts off the READ and WRITE signals from the microcontroller. Another signal (not shown in picture) asserted by the LCA informs the microcontroller that the LCA owns GPIB. If the microcontroller needs to read from or write to GPIB during a DMA-transfer it asserts a request signal telling the LCA it wants the bus. The LCA then releases the DACK signal and, in effect, hands over control to the microcontroller.

Memory map

The memory in MUG is divided in two parts. The ROM contains most of the microcontroller code for operating the MUG system, and the other half is RAM. In this RAM area there are addresses used for accessing the GPIB controller and the LCA. As can be seen in fig. 5 below there also is at least 1kbyte of internal RAM in the microcontroller (H8/338 has 2kbyte). To use the internal RAM it is required that the RAME bit in the control register of the microcontroller is set. The intentions are that some internal RAM can also be used for running occasional code, i.e programs that are downloaded to the MUG-system to be run for a special task. Programs in this area will execute faster because several clockcycles are gained when the microcontroller avoids fetching data externally.

The GPIB controller uses eight register positions accessed at the memory addresses FF80-FF87. This fact determines the granularity in the decoding of the address bus. All address bits but the three lowest are used in the decoding process. This means that it is possible to decode the memory space in blocks of eight addresses.



* For H8/336 and H8/337 the addresses F780 to FB7F is reserved and cannot be used

Fig. 5: Memory map for the MUG-system

Software

Another vital part of the MUG-system is software. The software for the system is divided into two main groups: code for the microcontroller and code for the LCA. In both cases there is the possibility to write software in high-level languages. Software for the microcontroller is preferably written in the C language. Low-level routines, for initialising the system and managing the system, have been written and tested altogether. Some routines for self-tests, especially the memory, have been written too. All of these routines have been put together to a set of MUG libraries. Each library includes routines for different parts of the system.

LCA designs are best described in the VHDL language. VHDL is not only used to describe the functionality but also to simulate it. The VHDL code could also be used to synthesize the design. Using commercial software tools, VHDL routines are written and assembled to aid the development of the logical functions needed. Another efficient way to construct LCA designs is to use schematic capture, i.e. to generate code from a schematic of the desired logic.

The purpose of building routine libraries is to simplify the adaptation of MUG to any given application. With time this library will grow as one is designing with MUG. The time for implementing an instrument or system based on MUG will then decrease.

Applications

The NPOND system

The new POND (*) (NPOND) system is a GPIB based data acquisition system intended for nuclear experiments. One of the vital parts in this system is the data acquisition module (DAM). The DAM is designed using a MUG board, and will therefore be described in some more detail. To gain further insight of the DAM, a short overview of the NPOND system is given, cf. fig 6.

In nuclear physics experiments for which NPOND was designed, there are detectors measuring different physical quantities. These quantities are digitized, using spectroscopic ADCs, and are then made available to the data acquisition module (DAM). The detector quantities are also used by the experimental trigger (ET) to determine if an interesting event is present and if so to trigger the ADCs and the DAM. When the ADC's have completed their conversion, the DAM transfers the data from the ADCs to the mass storage device (MSD), via the GPIB bus. During the conversion and the transfers, the DAM inhibits the ET.

The operator of the experiment controls the experimental equipment via the experimental controller (ECO). The ECO is a PC or MAC based man-machine interface (MMI). Via the ECO, commands can be sent to the DAM or the MSD. Typical commands would be start/ stop experiment, rewind tape etc. The ECO should also detect requests from any device that needs service while the experiment is running. Typically this will happen when the MSD is full of data.

Another facility available for the operator is the SPY device, which is used by the operator to display packets of experimental data while the experiment is running.

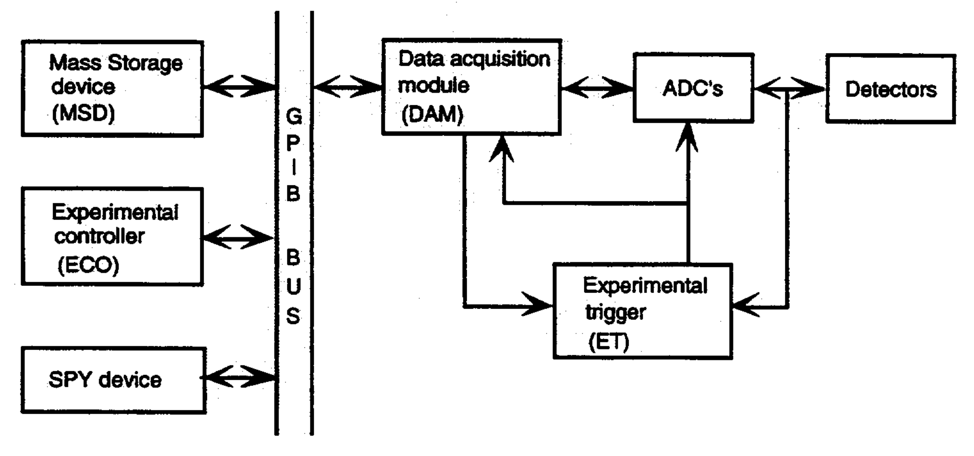
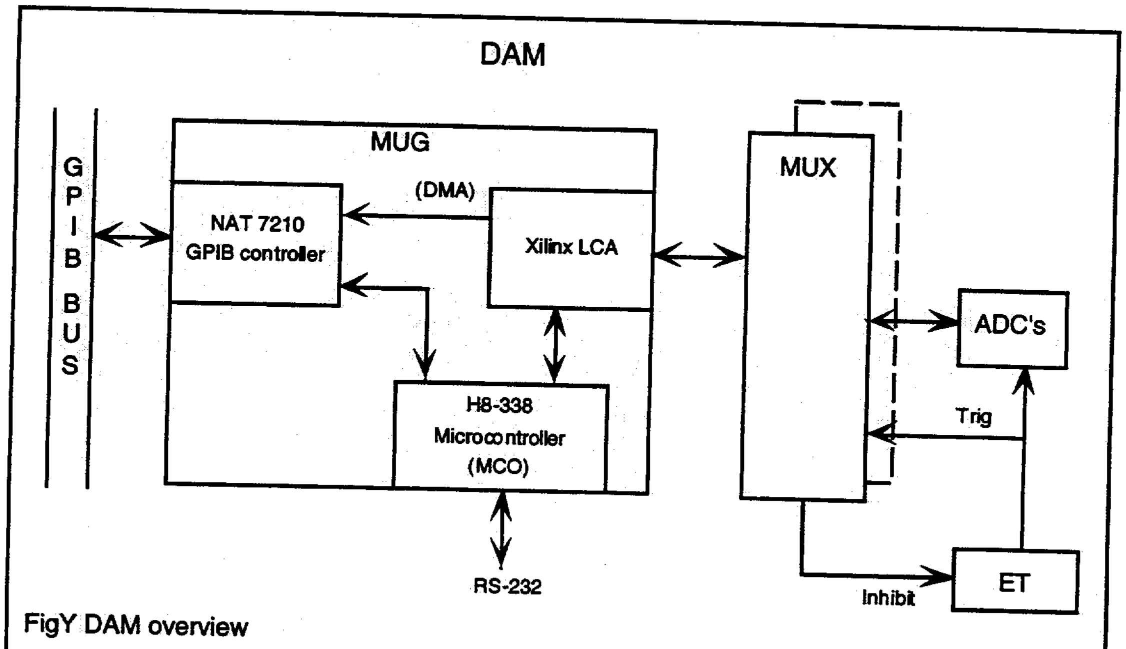

Mass Storage device (MSD)
Experimental controller (ECO)
SPY device
GPIB BUS
Data acquisition module (DAM)
ADC's
Detectors
Experimental trigger (ET)

Fig. 6: NPOND, system overview

(*) POND = Parallel transmission Of Nuclear Data

The heart of the DAM is a multi functional GPIB interface (MUG) board. To the MUG board it is possible to connect up to 8 multiplexor (MUX) boards, each of which can interface upto 15 ADCs. A MUG board and at least one MUX board combines to a DAM, cf. fig 7.



FigY DAM overview

The development of the NPOND project was the pilot project for the MUG concept. In the predesign phase of the NPOND project we were looking for a versatile hardware platform, where the platform must meet the following requirements:

- A communication link with a moderate bandwidth
- A great number of digital I/Os for interfacing a large number of ADCs
- It must be flexible so that ADCs from different vendors could easily be adapted
- It should not limit system performance, i.e not introduce overhead so that the transfer rate is reduced

All of the above requirements, except the I/O, were met by the MUG concept. The I/O requirements are rather extreme and hardly available from any platform. this is due to the fact that the NPOND system should be able to accommodate upto 120 ADCs. This means that one needs approximately 600 I/O signals with the ADC bus structure chosen in NPOND. To expand the available I/O from the MUG board we designed the MUX board. The MUX board(s) are attached to a simple bus that is controlled from the LCA of the MUG board. With the controller function located in the LCA, the MUX board design was very simple and cost effective because we had just to implement simple multiplexer functions.

The main task of the LCA is to be a link between tha ADCs and the GPIB bus. It uses the DMA facilities to improve the data transfer rate, thereby eliminating bottlenecks which could have been present if the microprocessor would have been responsible for this task. Other functions implemented in the LCA makes it easy to use ADCs from different vendors. These functions are programmed by the microprocessor by commands sent from the ECO over the GPIB bus.

The LCA design is rather complex but development time was significantly reduced by using VHDL tools during the implementation process. As mentioned earlier, parts of the VHDL code (i.e DMA controller) could be used as library functions when designing other MUG applications.

As with the LCA design, parts of the software in the MUG concept were actually developed within the NPOND project, adding to the software library functions. Therefore it took a little bit longer to develop the software. The actual power of the MUG concept were found when a simplified SPY interface was developed. In that interface one MUG board was used. Its taskwas to listen on the GPIB traffic and transfer data onto an RS232 serial channel. In this case no extra hardware was needed, just the MUG board itself. All the necessary functions for GPIB communication were now available as library functions, thereby the SPY interface application was very simple to develop. The time for the development of the SPY interface was counted in days, instead of weeks or months.

The next step in the NPOND project is to develop a real SPY interface. In such an interface the LCA should read data from GPIB using DMA, store the data in a buffer, and then transfer data through the RS232 serial channel. In this step we will again see the power in the MUG concept, due to the fact that there are both a software library as well as an LCA VHDL library available.

Future developments

In the nearest future an upgrade the GPIB-controller in the MUG-system is planned. The DMA-mode of MUG could be made to work at higher bandwidth if one uses high speed GPIB. In this version we can transfer data in blocks synchronously. This is done by not using handshake in the transfer between devices on GPIB. Instead data is sent in bursts, giving the possibility of making faster data transfers.

There are no built-in limitations in MUG restricting its use to GPIB as a bus to interface to. The architecture of MUG is open and very flexible and therefore suitable for changing the bus controller. There are other kinds of buses to interface to, e.g SCSI, which is also very common in computer hardware today. In the future we expect to design a similar system and replace the GPIB with SCSI, or possibly ethernet.

Conclusions

A solution to shorten the turn around time when making new instruments has been presented. It has been demonstrated that with programmable logic an all-around interface adapted to some specified bus can be developed. In this case GPIB chosen, but other bus architectures are not excluded. As shown, the MUG system is very flexible, due to its possibility to change the functions of the application logic via GPIB and the system logic can be programmed during run time.

References

NAT7210 Reference Manual, National Instruments

H8/338 Series Hardware: Users Manual, Hitachi

The programmable logic data book, Xilinx 1994

Appendix

GPIB

The IEEE-488.1 bus , or GPIB, accommodates a variety of devices that operate in one of three basic modes. The *listener* only accepts data from the bus, the *talker* sends data over the bus and the *controller* assumes supervisory control by addressing other devices or granting permission to use the bus. Devices connected to the bus can embody any combination of these three types. However, only one talker or one controller (controller-in-charge, CIC) can be active at a time. In a GPIB system a System Controller is required. The SC is the only controller able to take control of the bus. Devices are assigned unique addresses in the range 0 to 30. Information on the bus is typically transferred between a talker and a listener, but any number of listeners is possible. GPIB employs a byte-serial, bit-parallel, negative-logic scheme. Three different groups of signals exist:

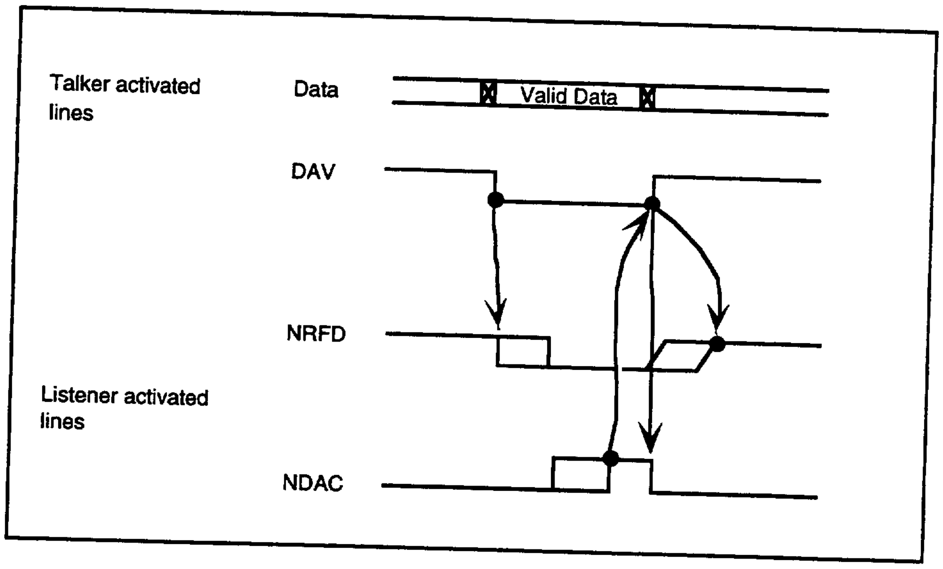
The eight *bidirectional data lines* carry a byte of address or data information or a command.

Three *byte-transfer-protocol lines* is used for handshaking.(DAV, NRFD, NDAC)

The five *interface-management lines* consist of the following signals:

ATN	(attention)- The controller pulls this line LOW, indicating that the information currently on the data lines is an interface address or command.
IFC	(interface clear) resets all interfaces to a particular known condition. This line may be used as a hardwired reset. This signal is asserted by the CIC.
SRQ	(service request)- A device requiring service calls on the CICs attention by asserting this wired-OR line LOW.
REN	(remote enable) allows remote control of a device - from front panel controls if the device is an instrument.
EOI	(end or identify) is asserted by the talker (denoting the end of a message-transfer sequence) or as a command from the controller (asking a device to identify itself in response to an SRQ). The device responds to the identify command by placing its address on the bus during the polling sequence.

Every byte transfer is accompanied by an asynchronous 3-wire handshaking on the byte-transfer-protocol lines. After making sure that the listeners are ready, the talker pulls DAV (data valid) LOW, thereby announcing valid information on the data lines. Acknowledging DAV LOW, the fastest listener pulls NRFD (not ready for data) LOW, signaling that it is busy with the current byte. When the slowest participating listener releases NDAC (not data accepted) which is held LOW since previous transfer, all listeners have accepted the byte. Sensing NDAC HIGH forces the talker to remove the message and pull DAV HIGH. Upon sensing this signal all listeners prepare for the next byte by pulling NDAC LOW. During this time the talker prepares the next byte for transfer and subsequently asserts DAV as soon as NDAC is LOW and NRFD is HIGH, cf. fig A1 below.


Talker activated lines
Data
Valid Data
DAV
NRFD
Listener activated lines
NDAC